Esercitazione
Reti Sequenziali Sincrone

Reti Logiche T

Ingegneria Informatica

Esercizio 1

Progettare in modo diretto una rete sequenziale sincrona che analizza continuamente **IN[7..0]**. Tali ingressi sono da considerarsi **sincroni** e sono validi solo quando il segnale **EN=1**.

La rete deve essere in grado di fornire due distinte informazioni, contando il verificarsi di due eventi:

1. Il carattere (valido) **attuale è pari** e il carattere (valido) **precedente era dispari** (uscita **PD[?..0]**)
2. Il carattere (valido) **attuale è dispari** e il carattere (valido) **precedente era pari** (uscita **DP[?..0]**)

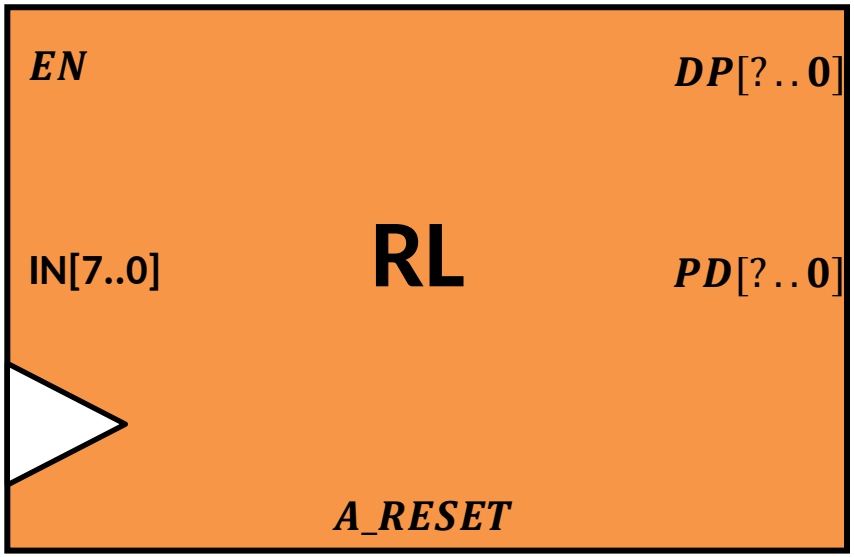
Entrambi i conteggi devono essere effettuati in aritmetica **modulo 16**, ossia i contatori devono partire da 0, incrementarsi di 1 fino a 15 e poi, all'incremento successivo, ripartire da 0.

La rete deve aggiornare le proprie uscite solo al termine del ciclo di clock nel quale si verifica uno degli eventi descritti.

E' presente inoltre un segnale, denominato **A_RESET**, che consente di inizializzare la rete all'avvio in modo asincrono. Al reset, si deve assumere di non aver visto nessun carattere valido in precedenza.

Esercizio 1

- **Sintetizzare la rete in maniera diretta riducendo al minimo l'utilizzo di risorse.**
- **Di quanti bit di uscita avrà bisogno la rete?**



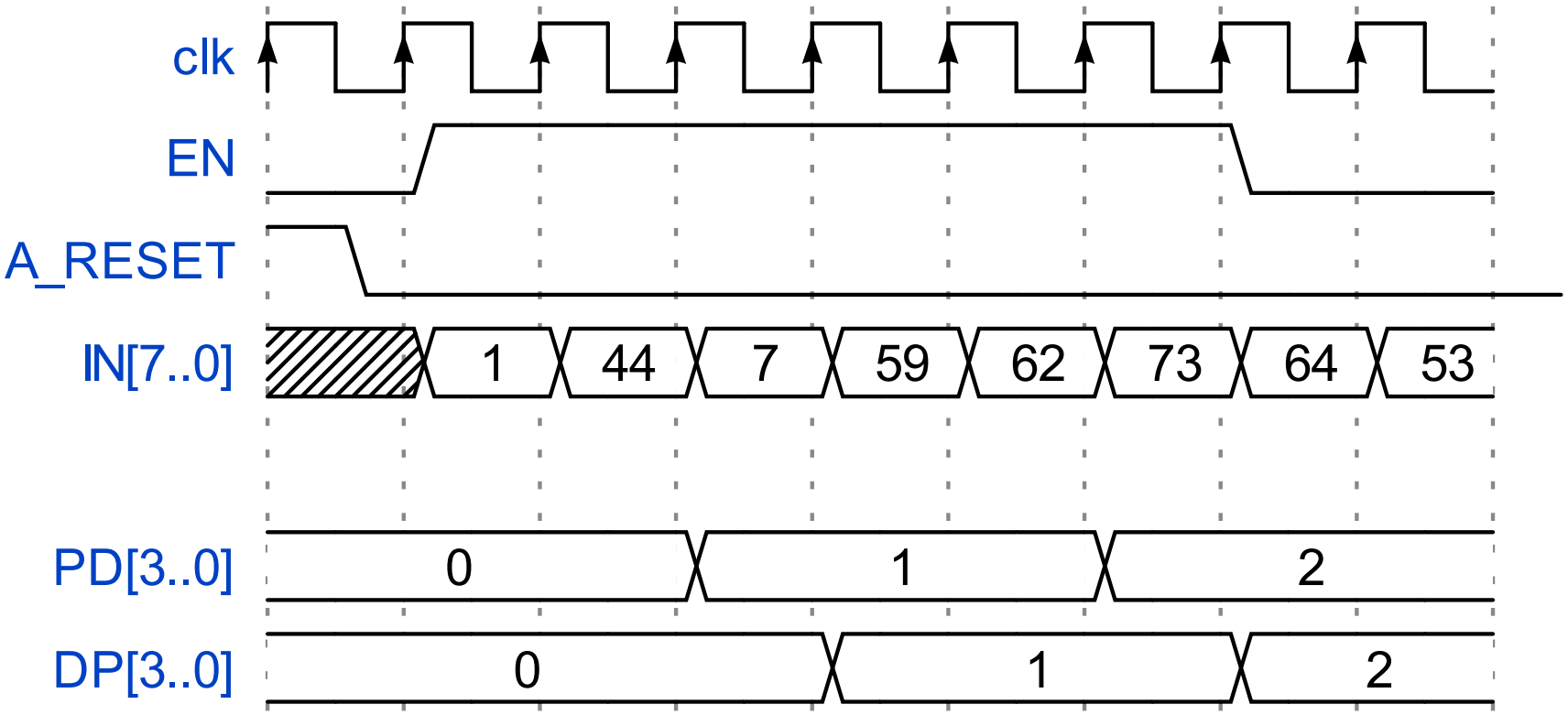
Esercizio 1 - Considerazioni

- La rete deve contare modulo 16 il verificarsi di due eventi distinti. Pertanto sia per l'uscita **PD** che per l'uscita **DP** avremo bisogno di 4 bit per codificare correttamente i conteggi.

- Nel caso in cui i due caratteri attuale e precedente siano entrambi dispari o entrambi pari nessuna uscita deve essere aggiornata.

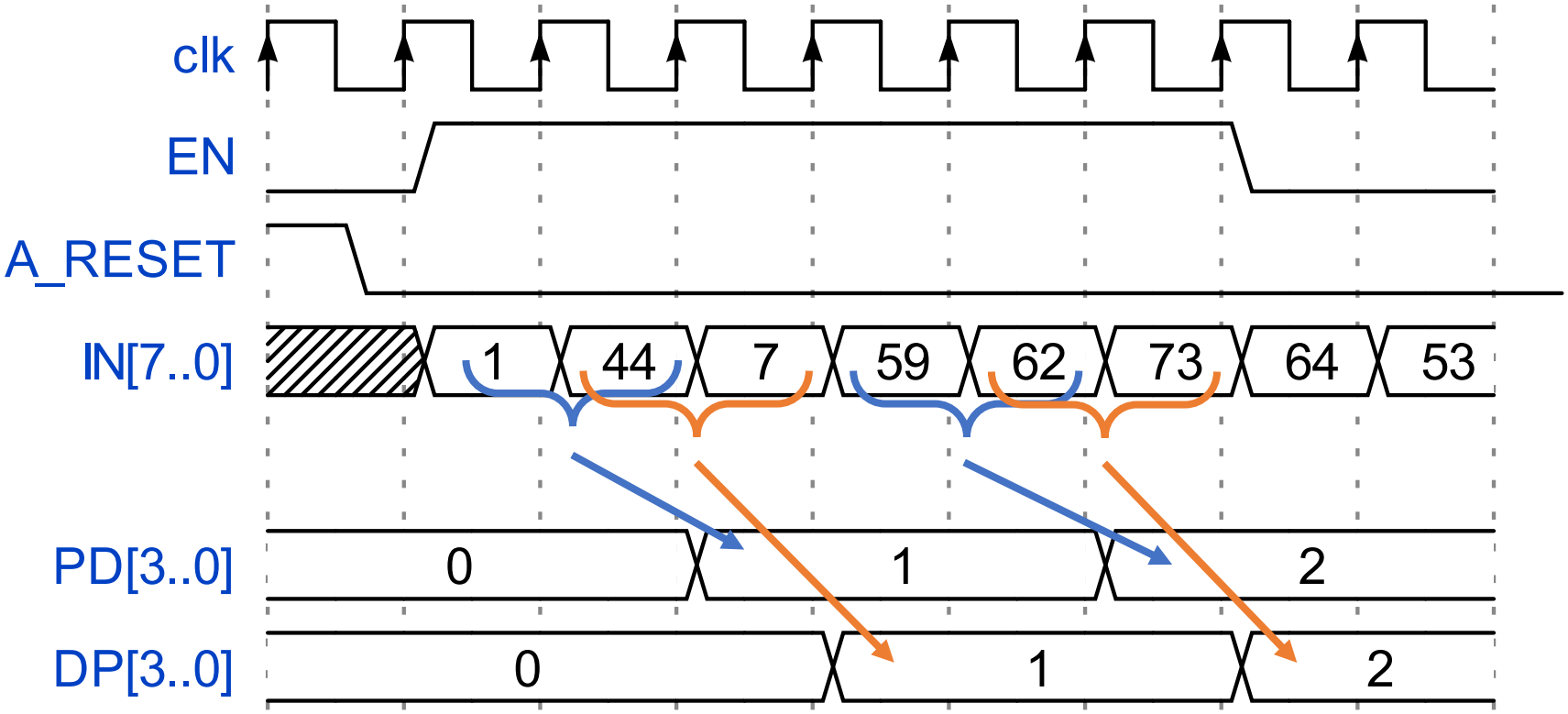
Esempio 1 – Forme D'onda

- Il comportamento che ci aspettiamo dalla rete è riassunto nelle seguenti forme d'onda.



Esempio 1 – Forme D'onda

- In blu coppie di numeri **Pari-Dispari**, in arancione coppie di numeri **Dispari-Pari**.



Esercizio 1 - Considerazioni

- Possiamo ottenere i due conteggi con due contatori modulo 16 comandando opportunamente i rispettivi *ENABLE*. I segnali di uscita **PD[3..0]** e **DP[3..0]** saranno ottenibili direttamente dalle uscite dei due contatori.

- I due contatori dovranno essere attivati solo quando si verificano in ingresso sequenze DP o PD.
  Per riconoscere queste sequenze avremo bisogno di **ricordarci** se il numero ricevuto in ingresso all'intervallo di clock precedente fosse pari o dispari.

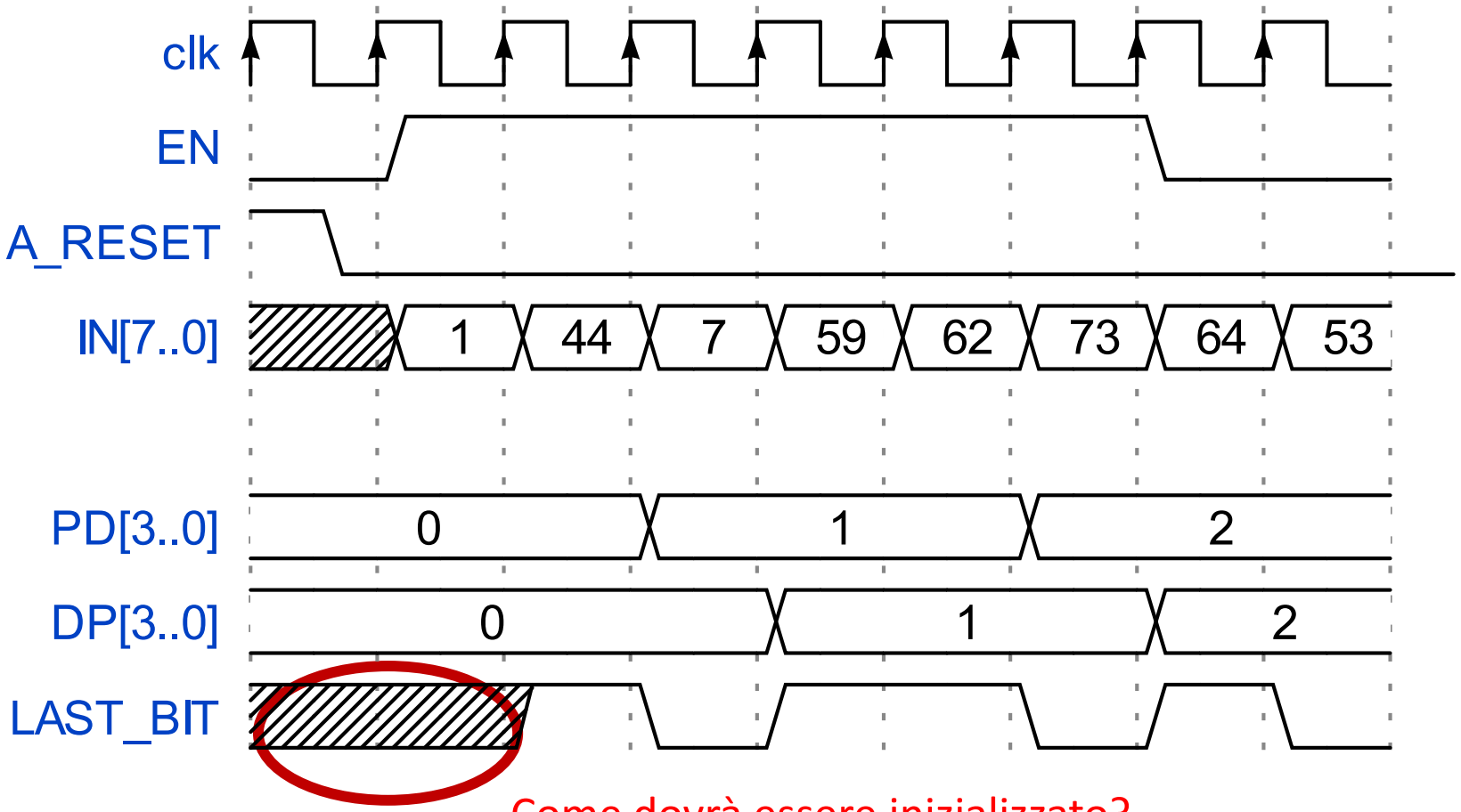
Esercizio 1 - Considerazioni

- Dovendo confrontare il carattere attuale con il precedente avrò bisogno di almeno un registro dove immagazzinare il penultimo dato ricevuto.

- Per ridurre le risorse utilizzate in termini di FFD, è possibile pensare di memorizzare solo il bit meno significativo del carattere in ingresso essendo possibile dedurre se l'intero carattere sia pari o dispari dall'analisi di questo unico bit.

Esercizio 1 – Forme D'onda

- Chiamiamo questo segnale **LAST_BIT.**



Come dovrà essere inizializzato?

Esercizio 1 - criticità

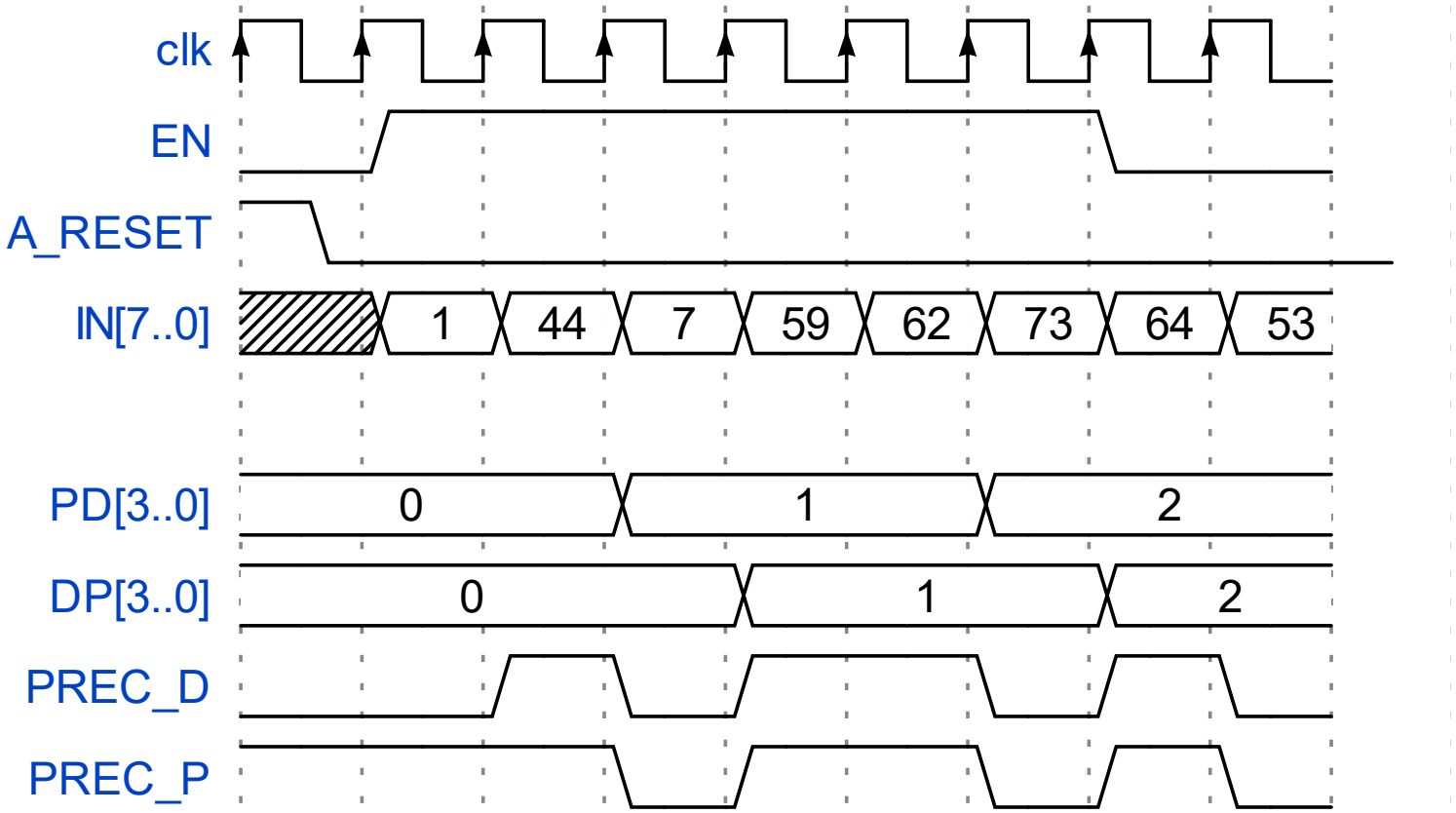
- Ne 0 ne 1 sono valori validi per l'inizializzazione di **LAST_BIT** perché assumerebbero un carattere precedente rispettivamente pari o dispari nella sequenza, quando in realtà non ci sono stati caratteri nella sequenza!!

- L'inizializzazione è una delle criticità della rete. Infatti, una volta inizializzata, la rete sarà in grado di fornire informazioni significative solo al termine dell'arrivo del secondo carattere valido.

- Tale problema può essere risolto con varie strategie: in questa soluzione dividiamo **LAST_BIT** in due segnali **PREC_P** e **PREC_D**. Entrambi memorizzeranno l'ultimo bit del carattere precedente, ma saranno inizializzati in maniera diversa in modo da evitare conteggi non voluti prima della ricezione del secondo carattere significativo.

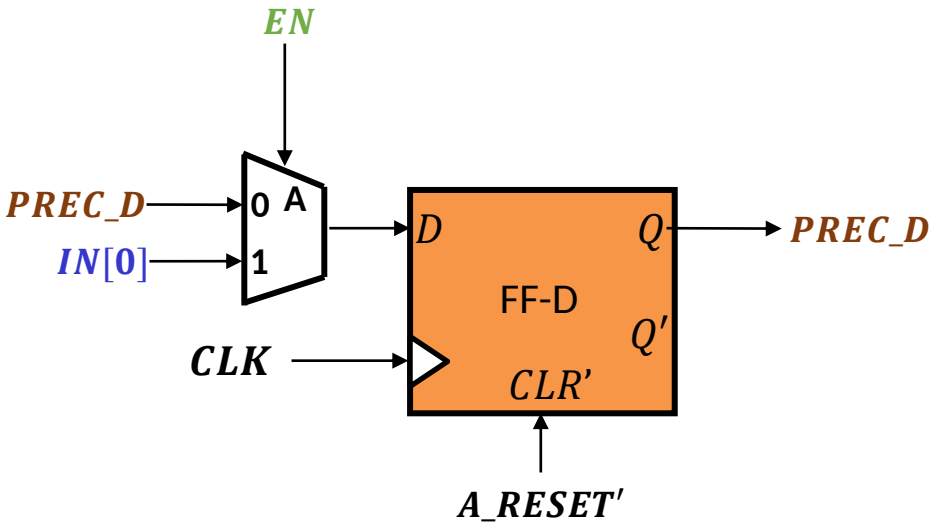
Esercizio 1 – Forme D'onda

- **PREC_D** verrà utilizzato per riconoscere sequenze **PD** e verrà inizializzato a 0 (bit precedente nella sequenza pari)
- **PREC_P** verrà utilizzato per riconoscere sequenze **DP** e verrà inizializzato a 1 (bit precedente nella sequenza dispari)



Esercizio 1 – rete PD

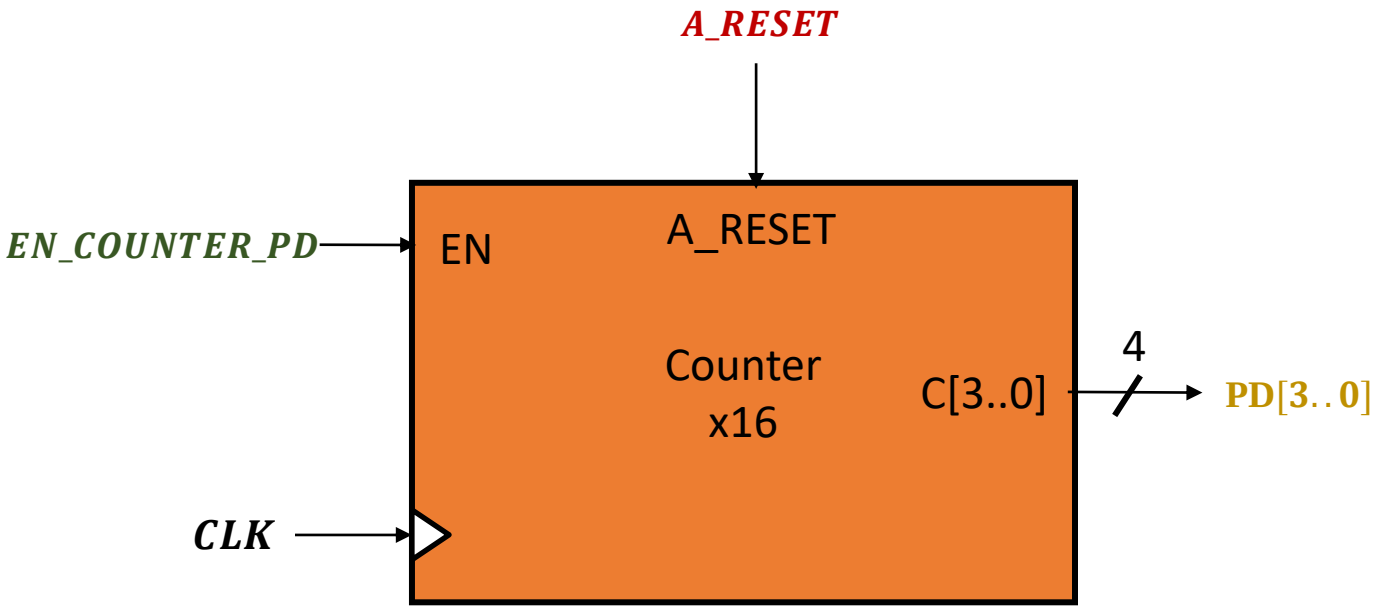
- Sintetizziamo una rete che memorizzi un carattere ogni volta che il segnale **EN** è valido e lo fornisca su **PREC_D**. Questo segnale codifica se il precedente carattere era dispari (i.e., **PREC_D**=1).
- Agiamo sull'inizializzazione della rete in modo da impostare **PREC_D** a zero quando viene asserito il segnale **A_RESET**.



Esercizio 1 – rete PD

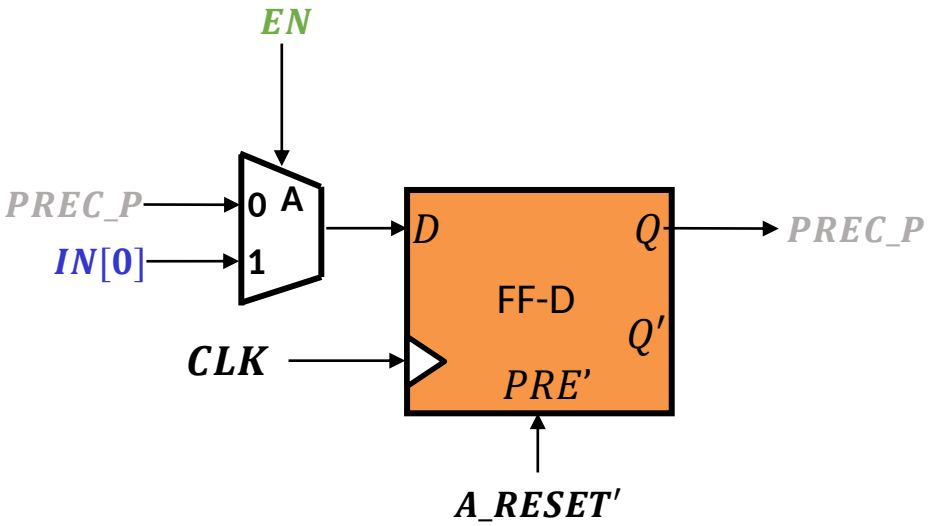
- Utilizzando il segnale **PREC_D** è possibile sintetizzare l'enable di un contatore modulo 16 che fornirà le uscite **PD[3..0]**.

$$EN\_COUNTER\_PD = EN \cdot PREC\_D \cdot IN[0]'$$



Esercizio 1 – rete DP

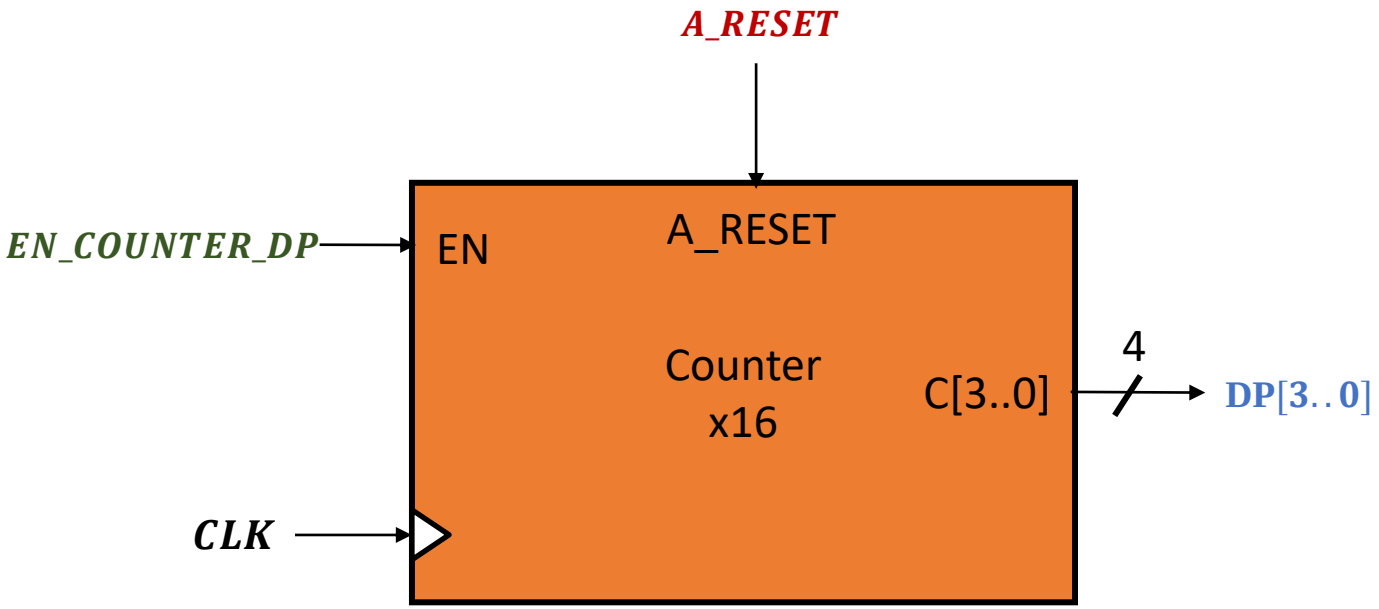
- Anche il segnale PREC_P memorizza l'ultimo bit ricevuto su **IN[0]**, ma deve essere inizializzato ad 1 e può essere utilizzato per decodificare se l'ultimo numero ricevuto era pari (i.e., PREC_P=0).
- Il segnale potrebbe essere ottenuto dall'uscita $\boldsymbol{Q}$ del FF-D precedentemente utilizzato per **PREC_D**, ma in quel caso avremo un comportamento errato all'inizializzazione (PREC_P deve infatti essere inizializzato ad 1). Per questa ragione utilizziamo un nuovo FF-D e colleghiamo il segnale $\boldsymbol{A\_RESET'}$ all'ingresso asincrono $\boldsymbol{PRE}$ per la sintesi di PREC_P.



Esercizio 1 – rete DP

- Utilizzando il segnale PREC_P è possibile sintetizzare l'enable di un contatore modulo 16 che fornirà le uscite DP[3..0].

$$EN\_COUNTER\_DP = EN \cdot PREC\_P' \cdot IN[0]$$



Esercizio 2

- Progettare in modo diretto una rete sequenziale sincrona che esegue continuamente l'accumulo **modulo 256** dei segnali d'ingresso **IN[5..0]** opportunamente elaborati come indicato in seguito.
- I segnali di ingresso, codificati come numeri senza segno e significativi solo quando **EN=1**, prima di essere sommati modulo 256 devono essere divisi per 4 se **DIV4=1** (lasciati inalterati se DIV4=0) o moltiplicati per 4 se **MUL4=1** (lasciati inalterati se MUL4=0). Si assuma che **DIV4** e **MUL4** non possano mai essere contemporaneamente ad **1**.
- Il valore dell'accumulatore deve essere mostrato sulle uscite **OUT[?..0]** e aggiornato ad ogni clock successivo alla ricezione di un carattere valido.
- Ogni volta che il valore accumulato risulta maggiore o uguale a 128, la rete deve invertire lo stato di un segnale **LED** di uscita **nel clock successivo alla ricezione del carattere valido.**

Esercizio 2

- Tutti i segnali, con l'esclusione di **A_RESET**, sono da intendersi **sincroni**. Il segnale asincrono **A_RESET** spegne il **LED** (**LED=0**) e azzera l'accumulatore interno della rete.
- Nelle divisioni non si consideri la parte frazionaria.

1. Quanti bit sono necessari per codificare l'uscita **OUT**?

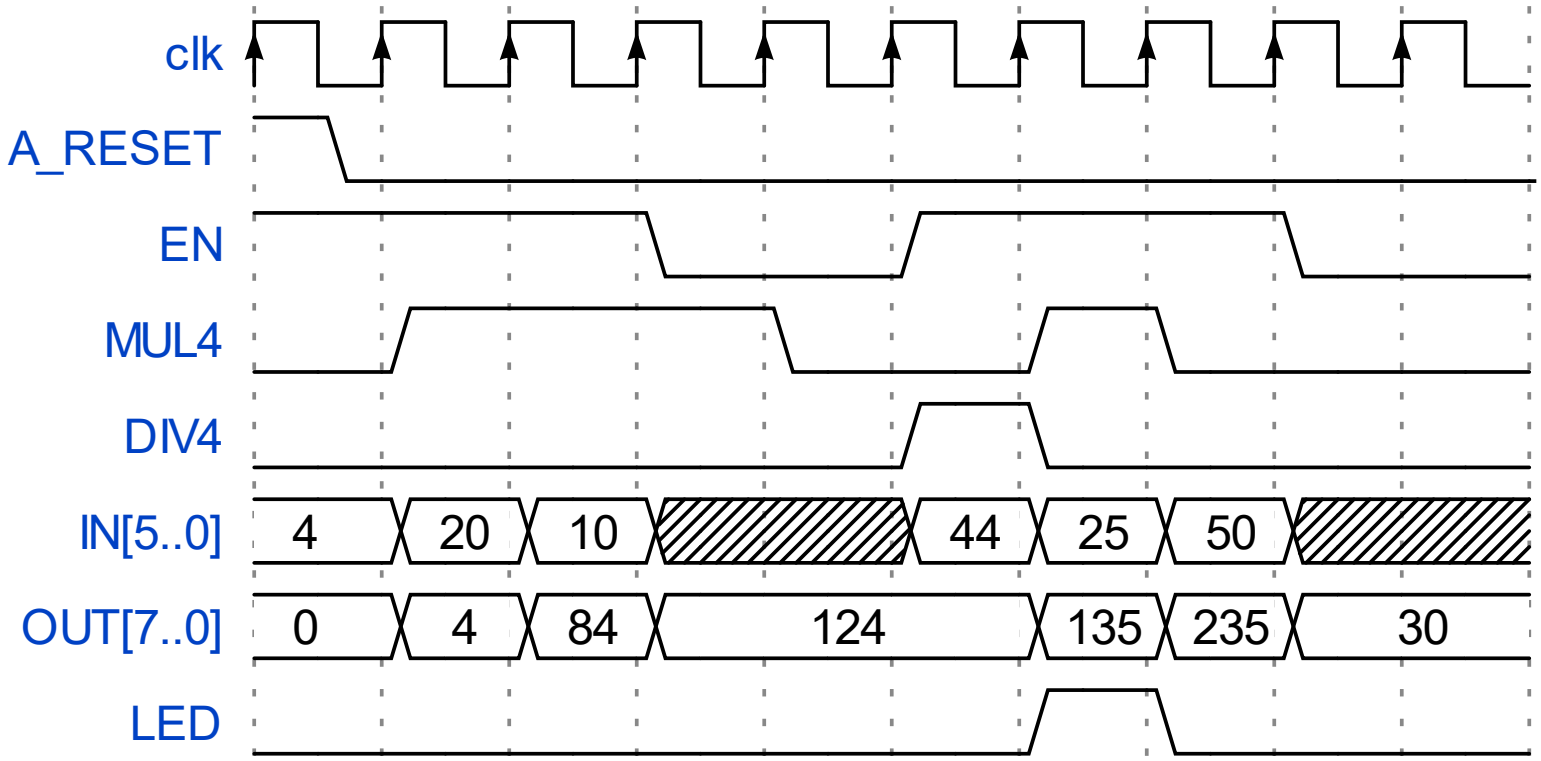
2. Sintetizzare la rete in maniera diretta riducendo al minimo l'utilizzo delle risorse.

Esercizio 2 - Considerazioni

- La rete deve accumulare modulo 256, saranno perciò necessari 8 bit sulle uscite per poter mostrare correttamente il risultato (**OUT[7..0]**).
- Per implementare un accumulatore avremo bisogno di un registro per memorizzare lo stato attuale della somma. Questo comportamento può essere implementato tramite un registro a 8 bit basato su **FFD** le cui uscite **OUT[7..0]** mostreranno il valore della somma esattamente un clock dopo la ricezione di un carattere valido.
- Lo stato del **LED** deve commutare un clock dopo la ricezione di un carattere valido contemporaneamente a ogni cambiamento dell'uscita **OUT[7..0],** se l'uscita risulterà maggiore o uguale a 128.
- Divisione e moltiplicazione per quattro possono essere ottenute shiftando i 6 bit di ingresso **IN[5..0]** rispettivamente a sx o dx di due posizioni quando i rispettivi segnali **MUL4** o **DIV4** valgono 1.
- I segnali di ingresso sono già tutti sincronizzati con il clock ad eccezione di **A_RESET.**

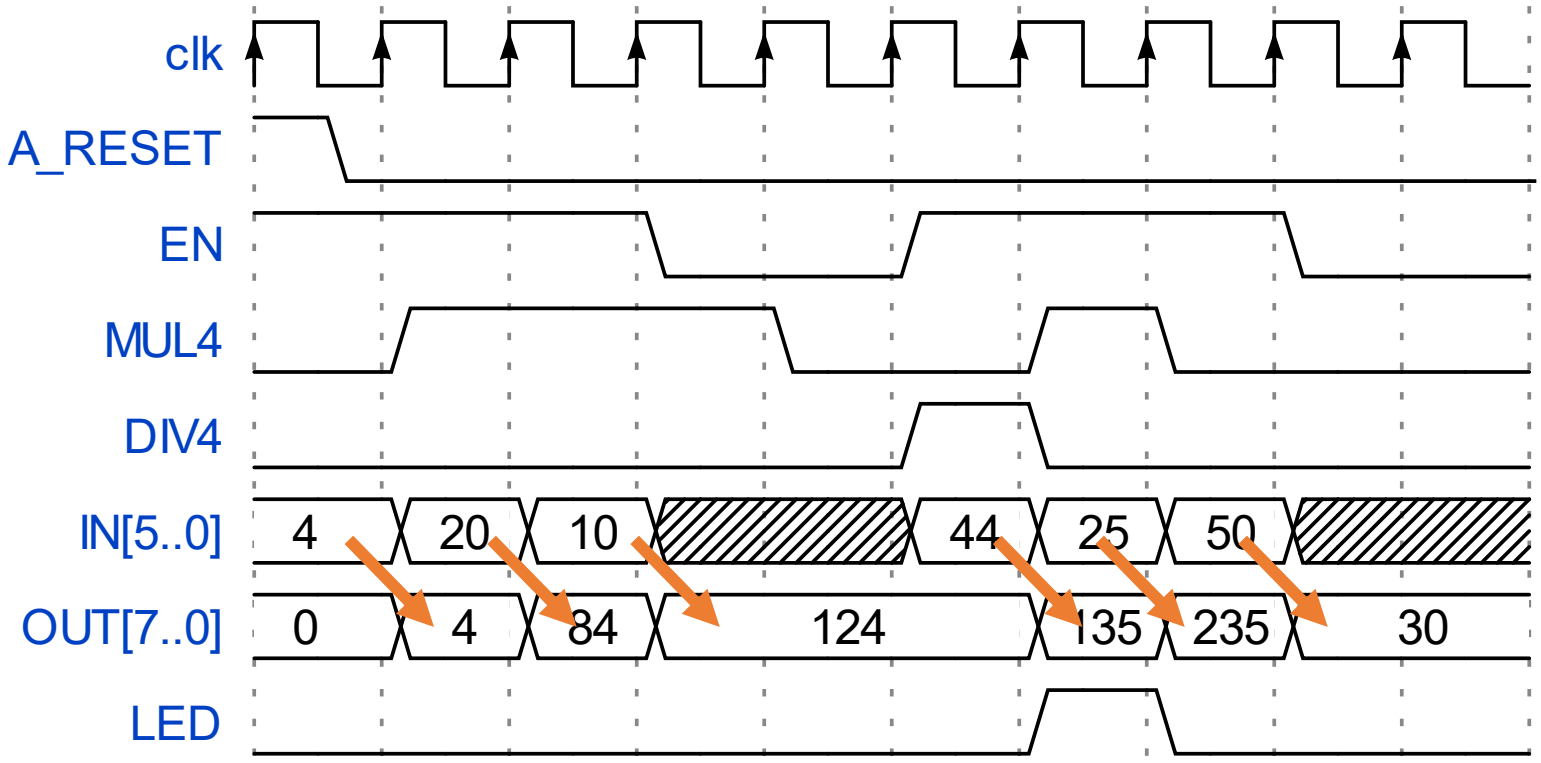
Esercizio 2 – Forme D'onda

Il comportamento che ci aspettiamo dalla rete è riassunto nelle seguenti forme d'onda



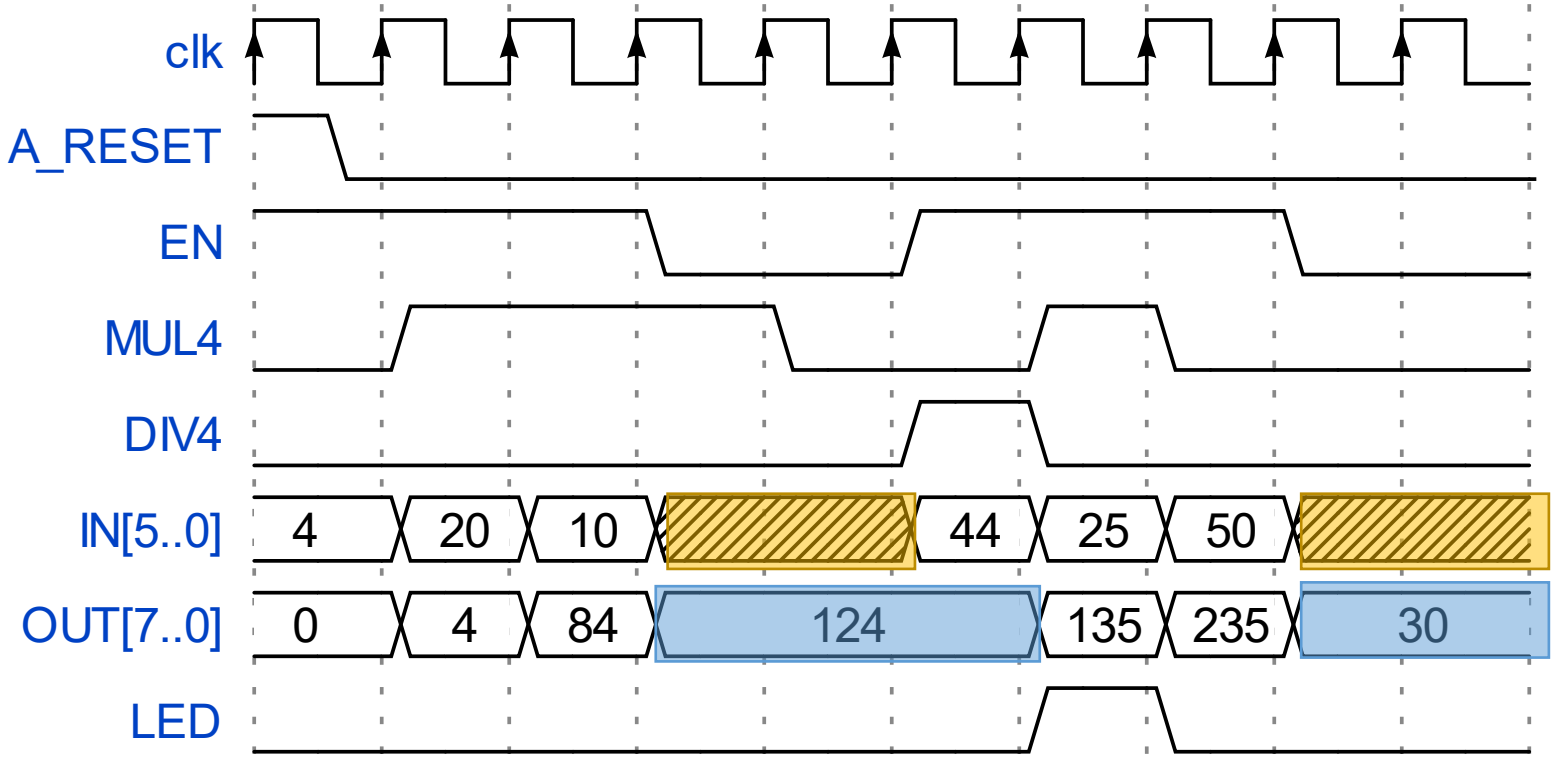
Esercizio 2 – Forme D'onda

- Le uscite **OUT[7..0]** devono essere aggiornate al clock successivo alla ricezione di un carattere valido (**EN**=1).



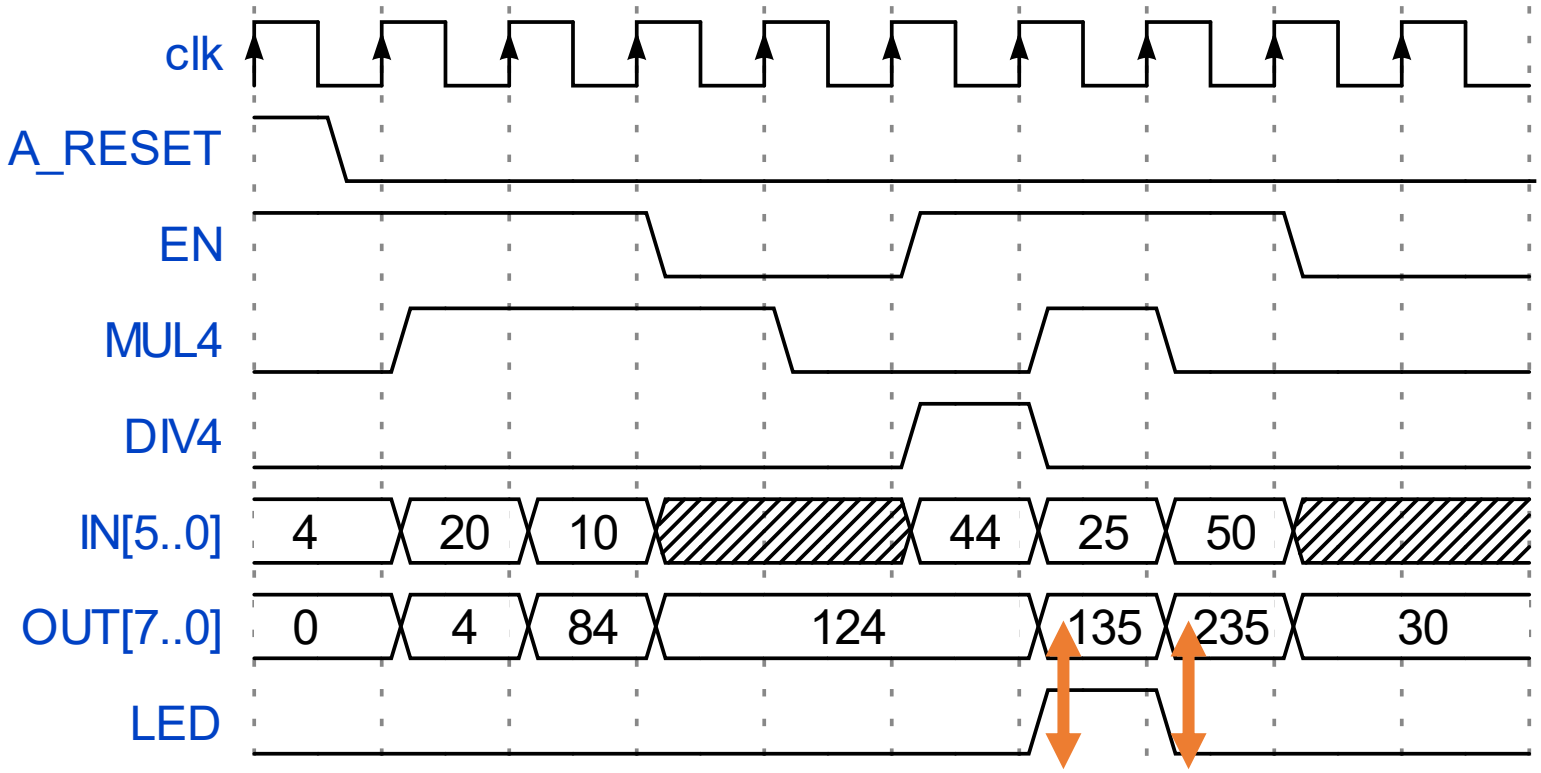
Esercizio 2 – Forme D'onda

- Gli ingressi **IN[5..0]** non sono significativi quando il segnale **EN**=0 (in giallo nelle forme d'onda sottostanti).
- Allo stesso modo le uscite **OUT[7..0]** devono essere mantenute costanti fino al clock successivo alla ricezione del successivo carattere valido (in celeste nelle forme d'onda sottostanti).



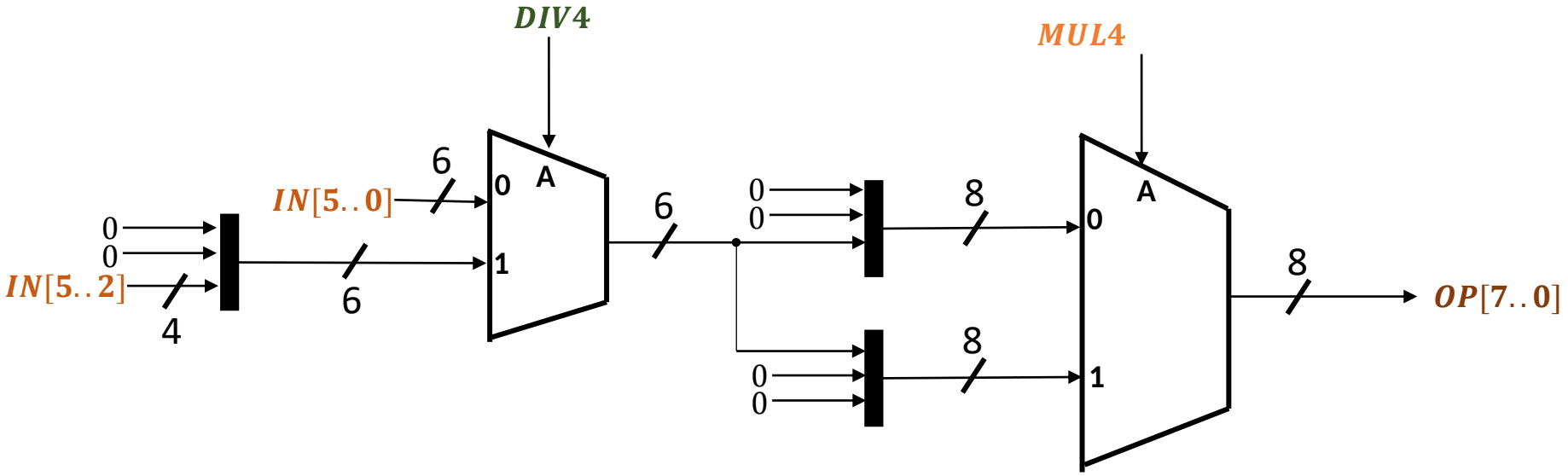
Esercizio 2 – Forme D'onda

- Il segnale **LED** deve commutare di stato ogni volta che le uscite **OUT[7..0]** assumono un valore >128.
- **ATTENZIONE: LED** deve commutare di stato in maniera sincrona con **OUT[7..0].**



Esercizio 2 – MUL/DIV

La moltiplicazione e divisione dei bit di ingresso può essere implementata in modo combinatorio (il valore deve essere disponibile nello stesso clock in cui arriva il dato) con due mux a due vie che portano i 6 bit di ingresso in una rappresentazione a 8 bit. Chiameremo gli 8 bit risultanti **OP[7..0]**.



Quando DIV4=1 shifto i bit di ingresso verso dx di due posizioni e aggiungo due bit a zero nelle due posizioni più significative.

Quando MUL4=0 aggiungo 2 bit a zero nelle posizioni più significative, non cambiando il valore dell'ingresso. Altrimenti li aggiungo nelle posizioni meno significative, shiftando gli input a sx di 2 posizioni.

Esercizio 2 - Accumulatore

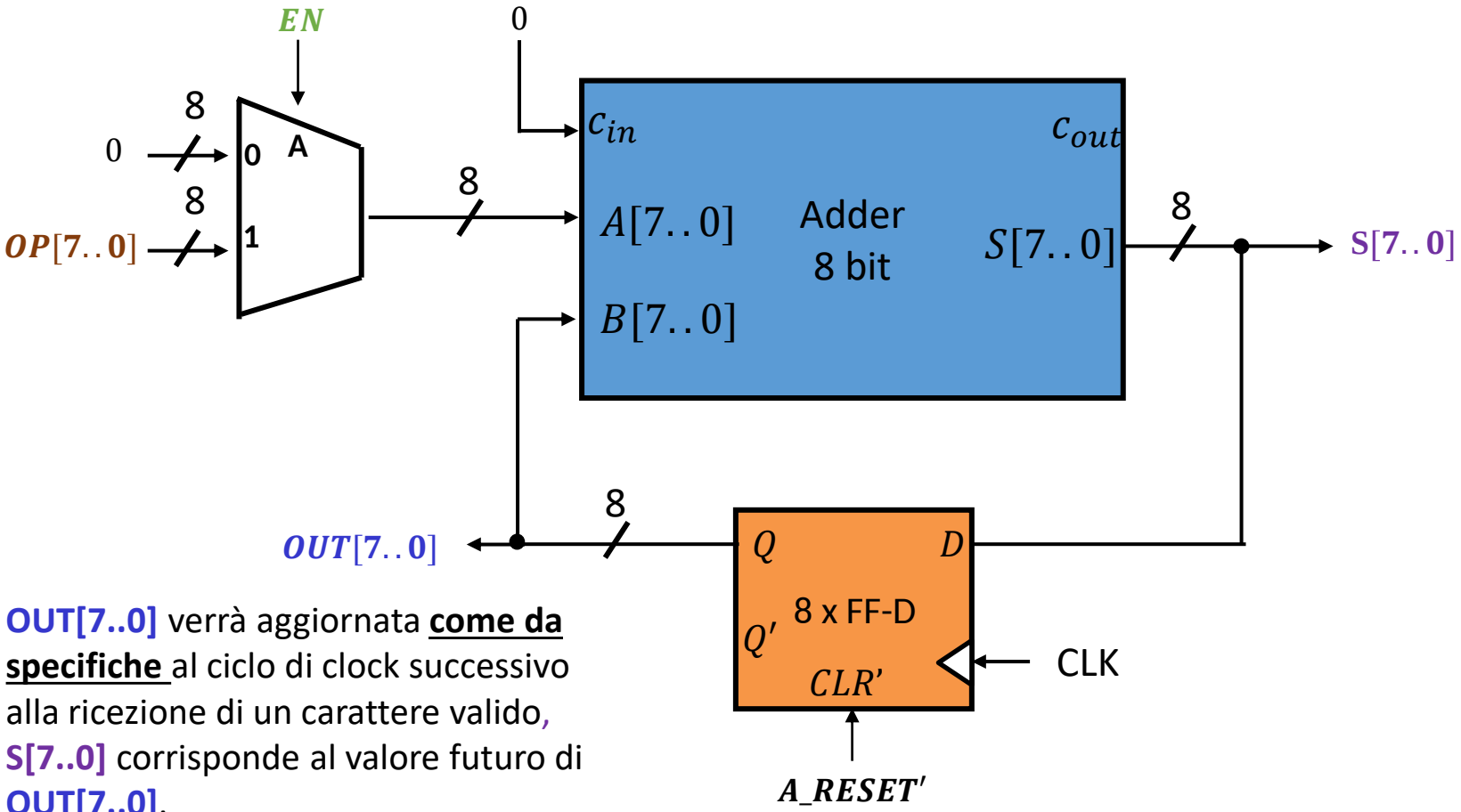
- Per accumulare la somma degli operandi ricevuti in ingresso e mostrarla sulle uscite con un ciclo di clock di ritardo, avremo bisogno di 8 FFD.

- I FFD devono essere opportunamente resettati dal comando asincrono **A_RESET**.

- La somma può essere implementata usando un adder ad 8 bit che sommi al valore attualmente salvato nell'accumulatore il valore di **OP[7..0]** nel caso in cui il segnale di **EN=1.**

Esercizio 2 - Accumulatore

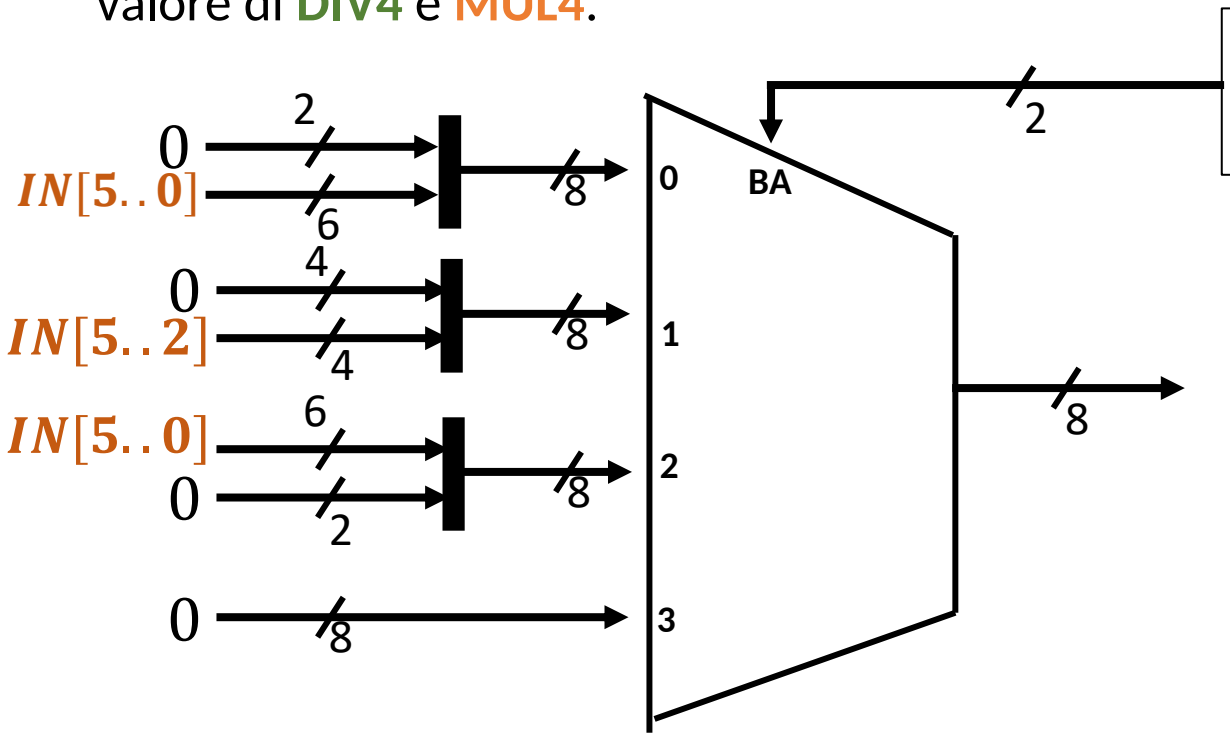
Il comportamento descritto in precedenza può essere implementato dalla seguente rete.



OUT[7..0] verrà aggiornata **come da specifiche** al ciclo di clock successivo alla ricezione di un carattere valido, **S[7..0]** corrisponde al valore futuro di **OUT[7..0]**.

Esercizio 2 – Minimizzazione risorse

- Una soluzione più efficiente consiste nell'utilizzare un unico multiplexer a 4 vie pilotato opportunamente in funzione dei segnali **EN**, **MUL4** e **DIV4**.
- Se **EN**=1 l'uscita del mux corrisponde a **OP[7..0]**, in base al valore di **DIV4** e **MUL4**. Se **EN**=0 l'uscita corrisponde a $(00000000)_2$ indipendentemente dal valore di **DIV4** e **MUL4**.

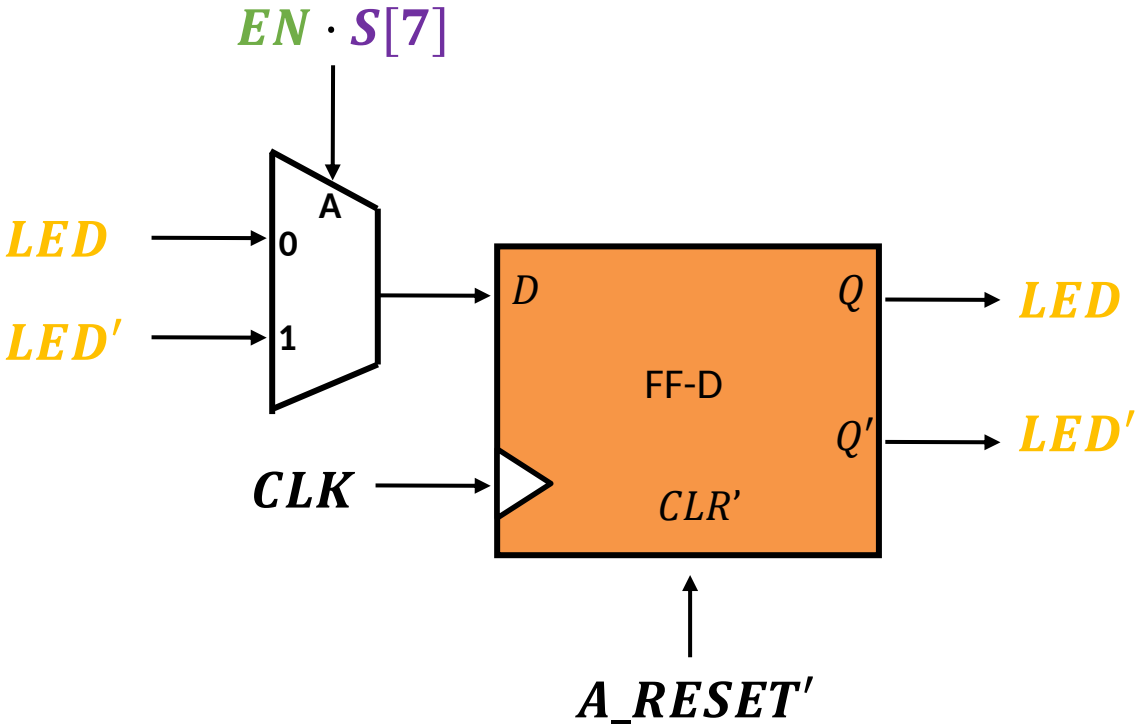


BA	SIGNIFICATO
00	EN=1, MUL4=0, DIV4=0
01	EN=1, MUL4=0, DIV4=1
10	EN=1, MUL4=1, DIV4=0
11	EN=0 (MUL4 e DIV4 non possono essere entrambi 1)

Esercizio 2 - LED

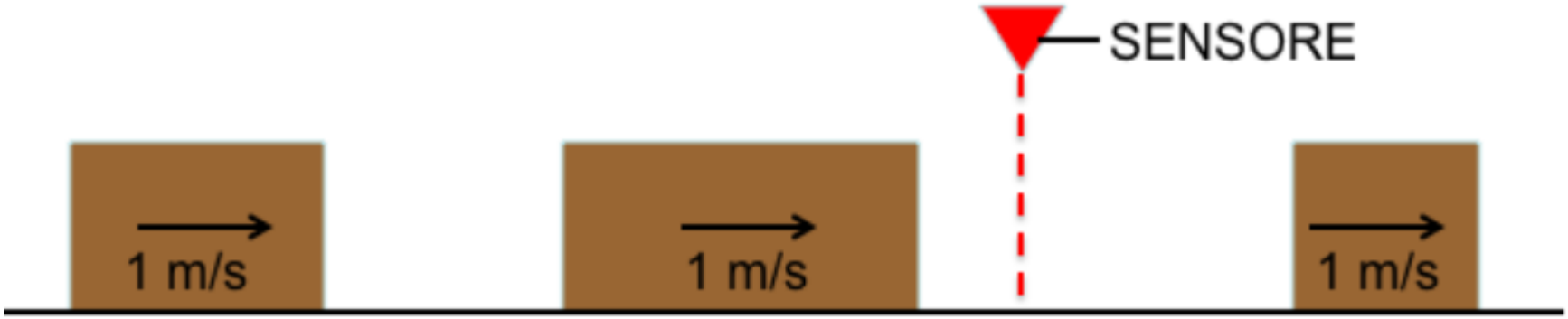
Possiamo sintetizzare la rete per l'inversione dell'uscita **LED** utilizzando un FFD. Il segnale deve cambiare di stato quando la somma presente nel prossimo clock sulle uscite **OUT[7..0]** è maggiore o uguale di 128, ovvero ogni volta che **S[7]**=1 e **EN**=1.

E' possibile sostituire il MUX con l'EXOR di LED e $EN \cdot S[7]$, come visto nel caso del contatore x4.



Esercizio 3

- Utilizzando un clock a **1 KHz**, progettare in modo diretto una rete sequenziale sincrona in grado di misurare la lunghezza, **in mm**, di oggetti di dimensione inferiore a **1 m** che si muovono alla velocità costante di **1 m/s** su un nastro trasportatore.



Esercizio 3

- Il segnale **SENSORE**, **per sua natura asincrono,** generato da un dispositivo non in movimento, vale **1** quando è presente un oggetto nell'area monitorata e **0** in caso contrario.
- Al termine di ogni misura,
  - Il segnale di uscita **READY** deve assumere il valore 1 per segnalare che la macchina è pronta a ricevere un nuovo oggetto e deve essere portato a 0 **a partire dal periodo di clock successivo alla rilevazione di un nuovo oggetto nell'area monitorata.**
  - il risultato della misura dovrà essere mostrato sui **t** segnali di uscita **LENGTH[t-1..0]**. I valori assegnati a **LENGTH[t-1..0]** dovranno assumere il nuovo valore **nel periodo di clock successivo a quello in cui l'oggetto misurato esce dall'area monitorata.**
- Infine, un segnale denominato **A_RESET**, consente di inizializzare all'avvio in modo asincrono il sistema **quando non sono presenti oggetti sul nastro trasportatore**.

Esercizio 3

Si proponga una soluzione basata su un approccio sincrono con sintesi diretta.

- Indicare quanti segnali sono necessari per codificare la misura.
- Ridurre al minimo l'utilizzo di risorse.
- Qual è il margine di errore che possiamo garantire sulla misura e da cosa è causato, assumendo flip-flop con setup time e hold time pari a 0, ovvero ignorando problemi di metastabilità?
- Come potrebbe essere aumentata la risoluzione nella misura?

Esercizio 3 - Considerazioni

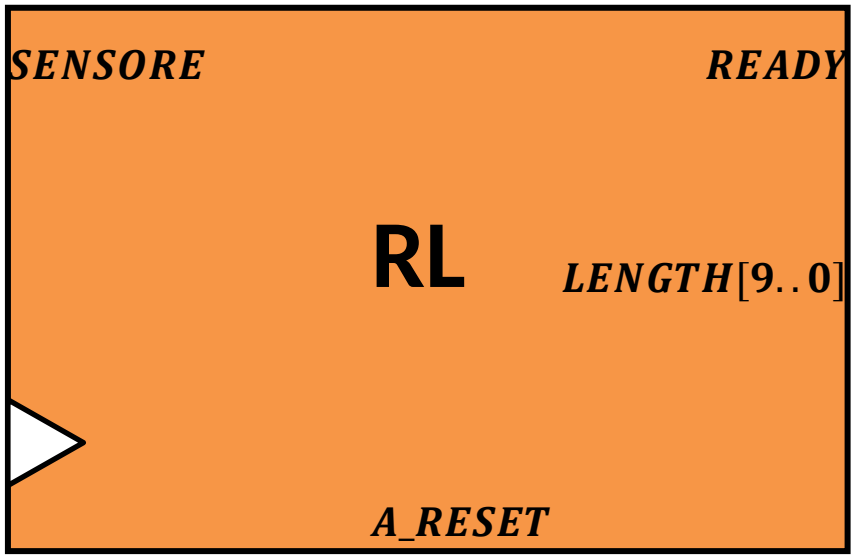
- La lunghezza dei pacchi può essere ottenuta misurando il tempo necessario ad attraversare l'area monitorata dal **SENSORE.** Per esempio un pacco di lunghezza **500mm** impiegherà **0.5s** a passare completamente sotto il sensore, ovvero **500** oscillazioni di clock considerando una frequenza di **1KHz.**
- La lunghezza massima di un pacco è **1m**, ovvero **1000mm**, che può essere codificata con 10 bit, quindi servono t=10 segnali di uscita **LENGTH**
- Per misurare la lunghezza dei pezzi in millimetri servirà quindi **un contatore a 10 bit (x1024).** Il contatore dovrà essere abilitato al conteggio quando il pezzo si trova sotto al sensore e resettato prima dell'inizio della nuova misura per essere pronti ad iniziare a contare all'arrivo del nuovo pezzo**. Serve quindi un contatore con ENABLE e RESET.**
- Il risultato della misura dovrà essere salvato in **un registro a 10 bit** per mantenerne costante il valore in uscita durante la nuova misura.

Esercizio 3 - Considerazioni

- Il segnale **SENSORE** **non è sincronizzato** con il clock, dovremo quindi sincronizzarlo.
- Dopo la sincronizzazione di **SENSORE**, una variazione da 1 a 0 del segnale sincronizzato **SENSORE_SYNC** indicherà la fine della misura.
- Il segnale **A_RESET** consente di inizializzare la rete.

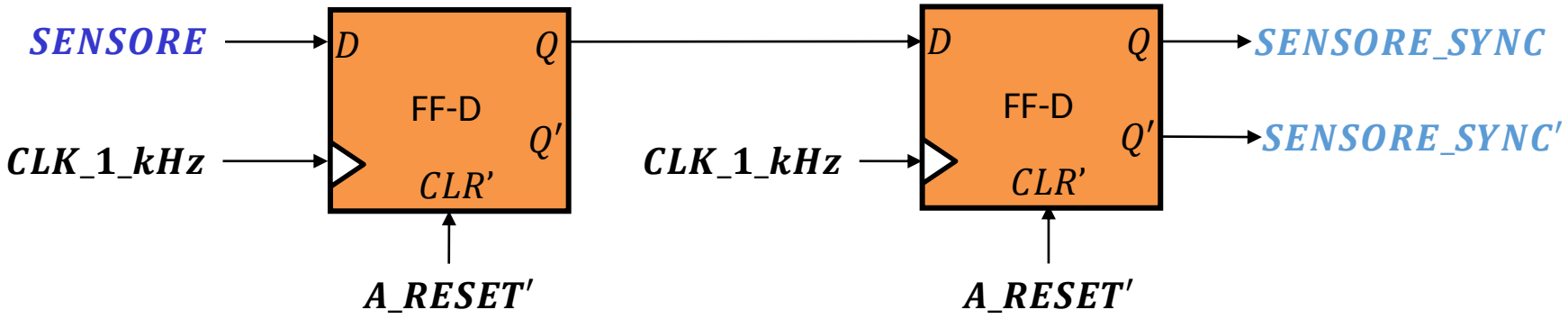
Esercizio 3 - Considerazioni

La rete nella sua interezza può essere rappresentata come un componente con 2 ingressi, 1 clock e 11 uscite.



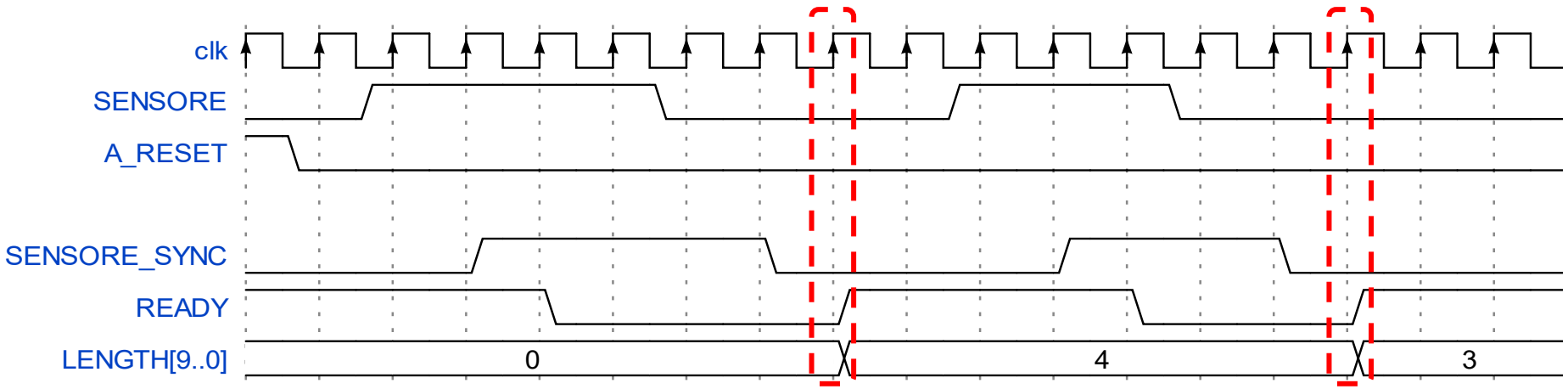
Esercizio 3 – Sincronizzazione

- Per prima cosa è necessario sincronizzare il segnale **SENSORE** con il clock a 1 **KHz.**
- La sincronizzazione può essere ottenuta mediante due **FFD**, chiameremo il segnale risultato della sincronizzazione **SENSORE_SYNC**.



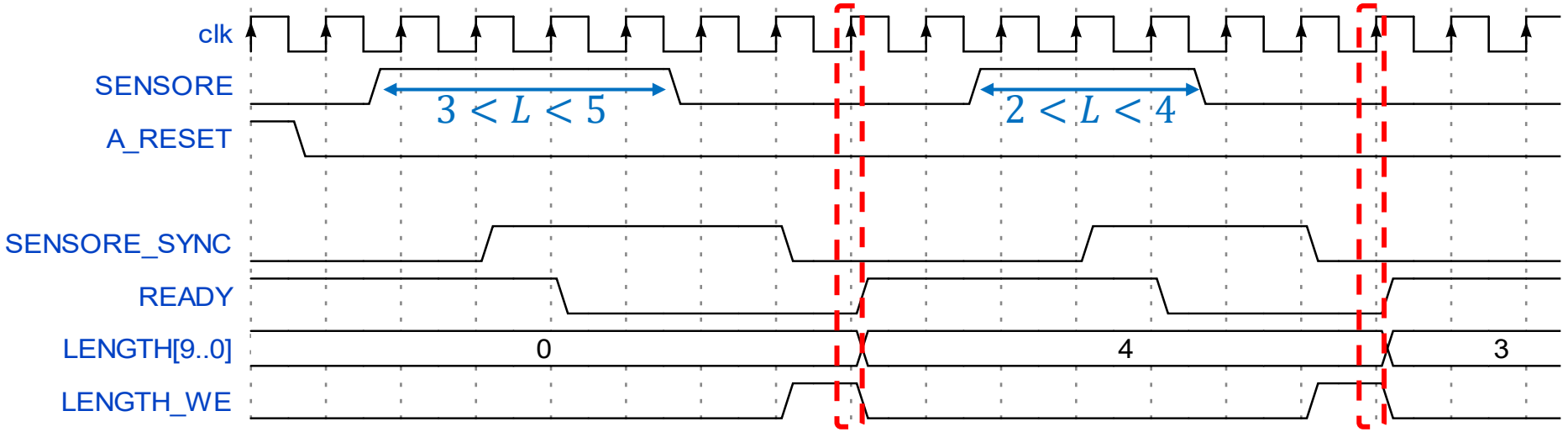
Esercizio 3 – Forme D'onda

- Il segnale **A_RESET** porta **LENGTH[9..0]** a $(0)_{10}$ e READY a «1»
- La fine di una misurazione può essere identificata dalla rete con il primo fronte del clock in cui **SENSORE_SYNC**=«0» dopo una misura (tratteggio rosso). Quello è l'istante di sincronismo in cui vengono aggiornate le uscite.
- Il segnale sincronizzato **SENSORE_SYNC** può essere usato anche per realizzare l'uscita READY richiesta: si può notare che READY è la versione ritardata di 1 clock di **SENSORE_SYNC'**



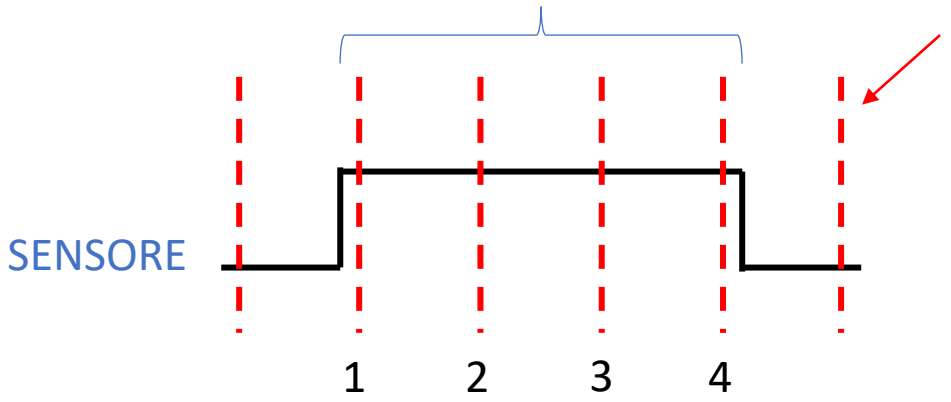
Esercizio 3 – Registri e precisione

- **LENGTH[9..0]** mantiene il suo valore fino alla fine della misura successiva.
- La sincronizzazione (ignorando problemi di metastabilità) introduce al massimo **un errore di misura di un clock**, **ovvero 1mm**, su dove inizia l'oggetto, e altrettanto su dove finisce.
- Per esempio se ho 4 periodi di clock con **SENSORE_SYNC**=«1», l'oggetto misurato avrà una lunghezza compresa tra 3 e 5 mm (vedi slide successiva).
- Per poter aggiornare il contenuto del registro **LENGTH** devo generare un segnale di write enable **LENGTH_WE**, che sia «1» per il periodo di fine misura.



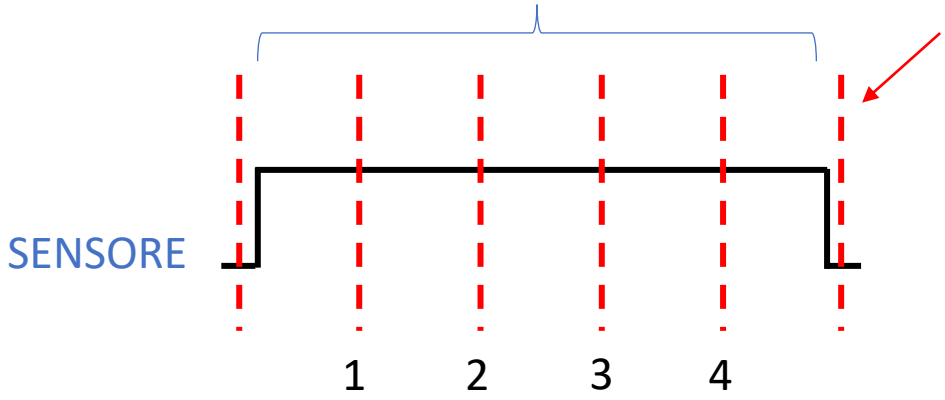
Esercizio 3 – Errore di misura

Oggetto di lunghezza 3.1 mm



Visto che SENSORE viene campionato solo al fronte di salita del clock, un oggetto di lunghezza di poco superiore a 3 mm può dare origine ad una misura di 4 mm.

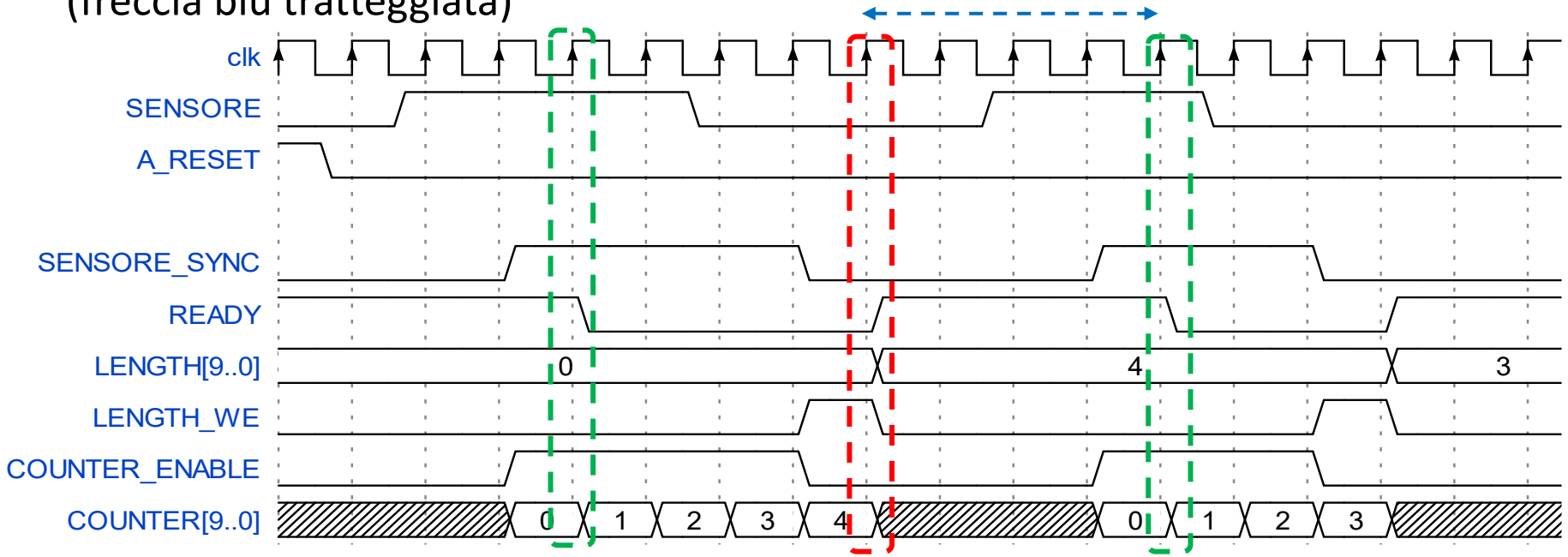
Oggetto di lunghezza 4.9 mm



Visto che SENSORE viene campionato solo al fronte di salita del clock, anche un oggetto di lunghezza di poco inferiore a 5 mm può dare origine ad una misura di 4 mm.

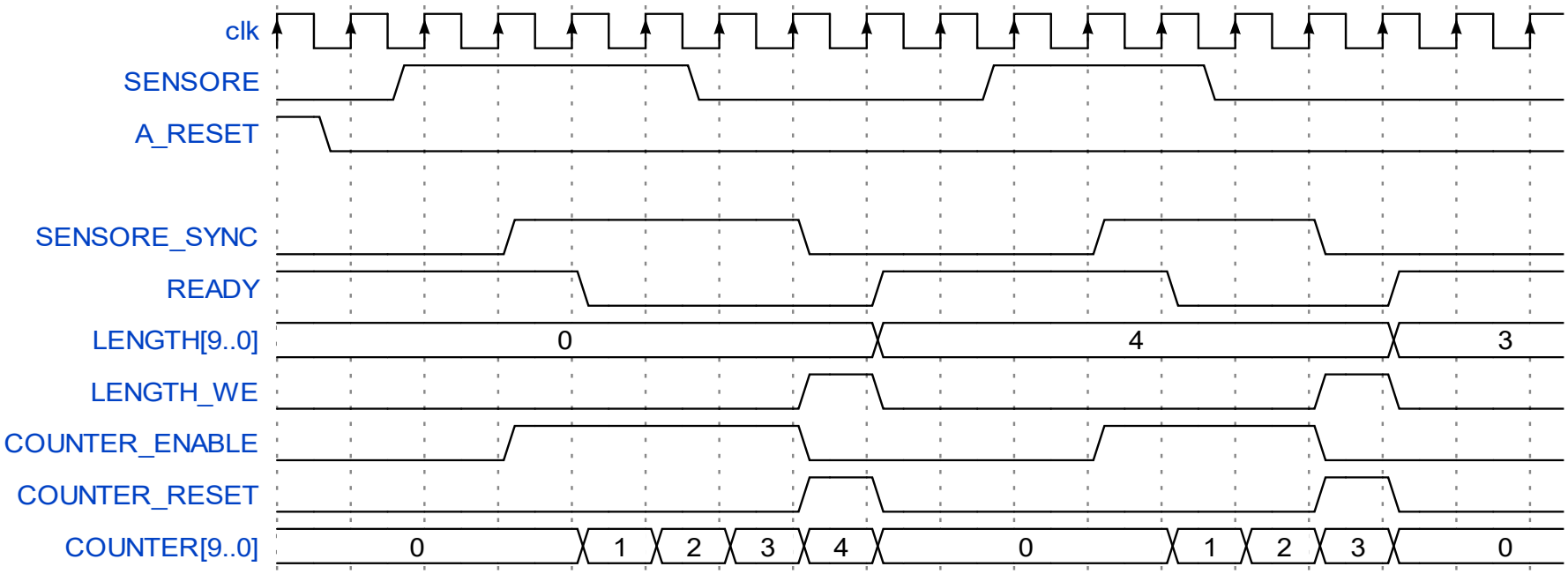
Esercizio 3 – Forme D'onda Contatore

- Per ottenere la misura richiesta, è necessario che lo stato del contatore segua l'evoluzione riportata in figura. Lo stato interno del contatore non è influente una volta che è stato salvato nel registro **LENGTH[9..0]**, purché si trovi in stato «0» quando inizia una nuova misura.
- Quindi, il segnale che abilita al conteggio il contatore (**COUNTER_ENABLE**) deve avere l'andamento riportato, che coincide con **SENSORE_SYNC**.
- Il segnale che resetta il contatore deve assumere almeno una volta il valore «1» tra la fine della misura precedente e prima dell'inizio della successiva (freccia blu tratteggiata)



Esercizio 3 – Forme D'onda Reset

- Non potendo prevedere quando inizierà la misura successiva decidiamo di **resettare il valore del contatore non appena termina un'operazione di misurazione**.
- Dalle forme d'onda possiamo notare che **COUNTER_RESET** = **LENGTH_WE**.

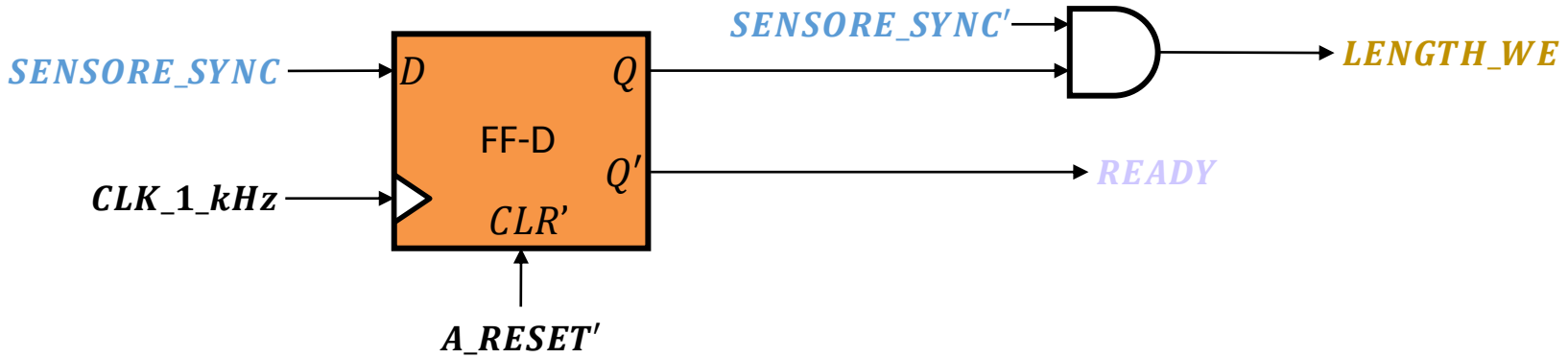


Esercizio 3 – LENGTH_WE & READY

Utilizzando **SENSORE_SYNC** è possibile sintetizzare il segnale **LENGTH_WE** che identifica la fine di una misurazione, ovvero fronte di discesa di **SENSORE_SYNC,**

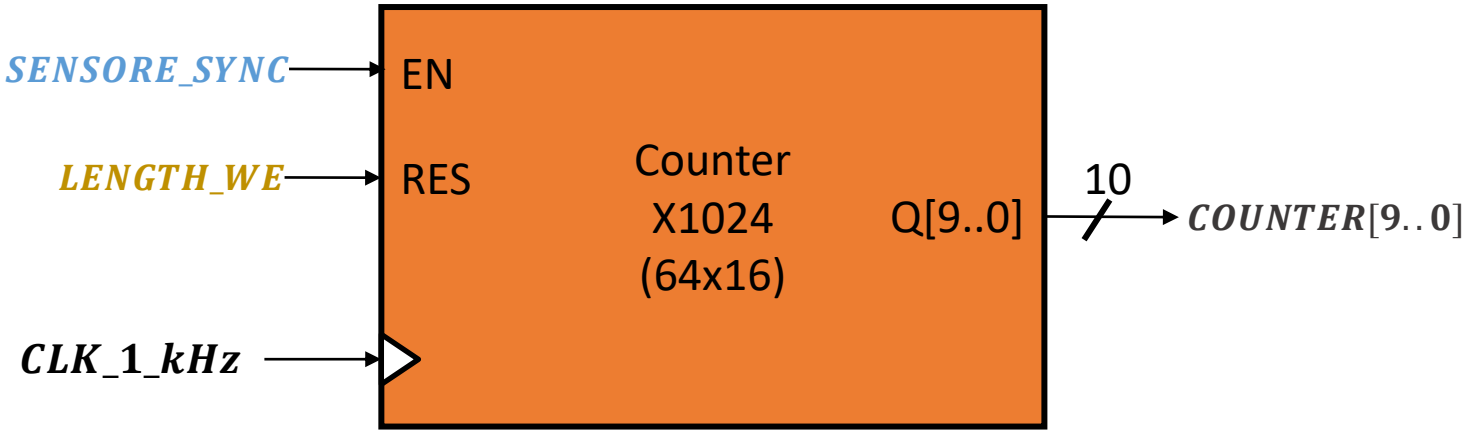
Per la sintesi abbiamo bisogno del valore assunto da **SENSORE_SYNC** all'intervallo di clock precedente. Il valore può essere campionato con un **FFD** (come visto a lezione, monoimpulsore 2).

Grazie al FF-D usato per realizzare lo stadio finale di un monoimpulsore, il segnale di uscita **READY**, che come già discusso è il complemento della versione ritardata di un clock di **SENSORE_SYNC**, è già disponibile tramite l'uscita Q'.



Esercizio 3 - Misurazione

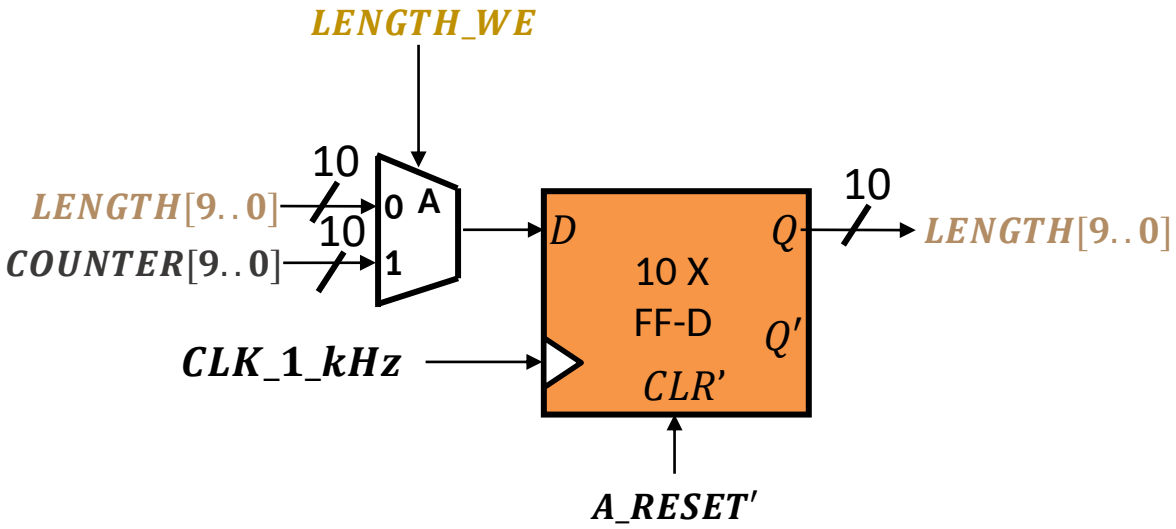
Il contatore può essere pilotato adeguatamente inviando **SENSORE_SYNC** sul pin di enable *EN* e **LENGTH_WE** sul pin di *RES*.



Si faccia riferimento alle slide di teoria per l'implementazione di un contatore x1024 con segnali di **EN** e **RES.**

Esercizio 3 - Memorizzazione

Infine dobbiamo memorizzare il valore del contatore al termine del transito dell'oggetto nell'area monitorata dal sensore in modo da fornire sulle 10 uscite le dimensioni dell'ultima misurazione in maniera stabile.



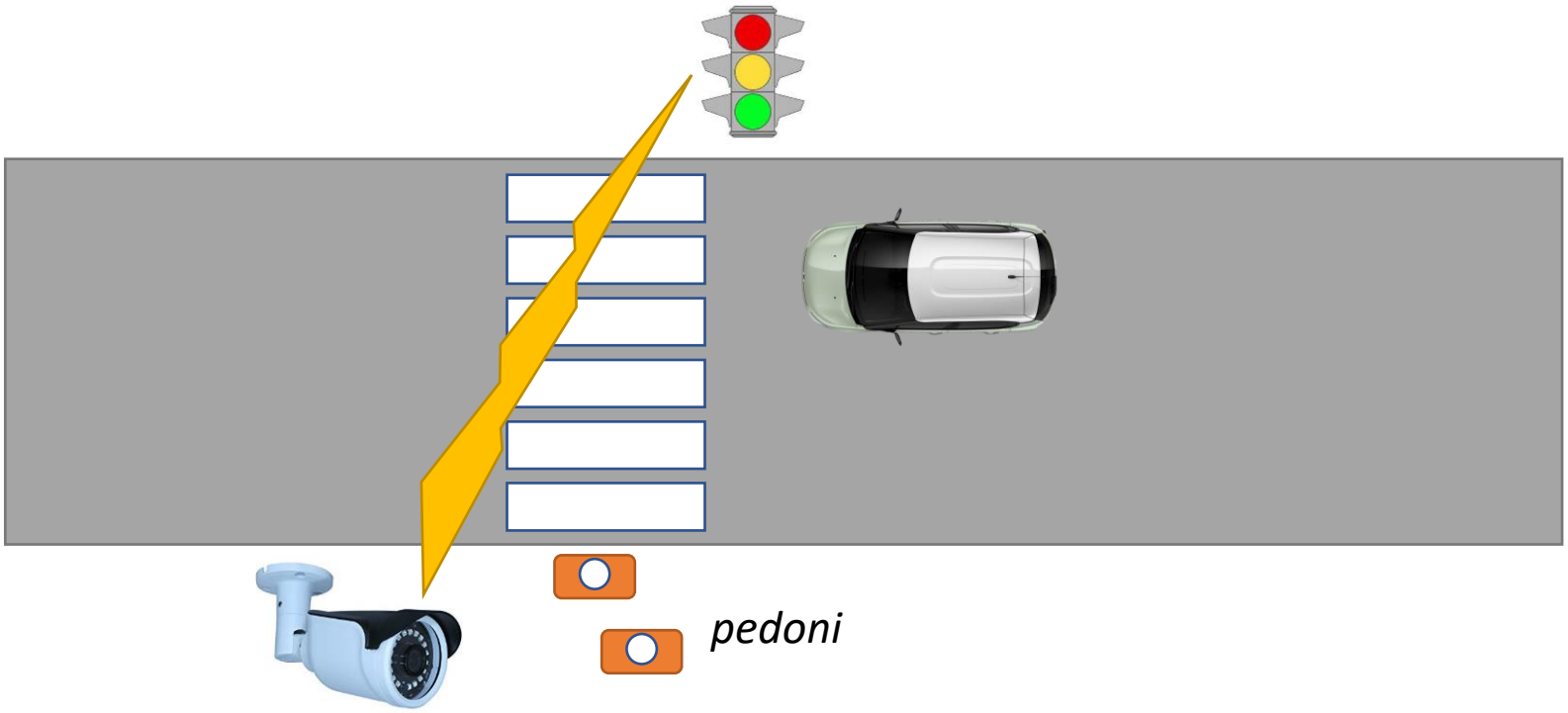
Esercizio 3 – Cambio di Risoluzione

- Il sistema descritto ha una risoluzione massima di **1 mm** dato che in un'oscillazione di clock il pacco sotto il sensore si muove di esattamente **1 mm**.

- Se volessimo aumentare la risoluzione dovremo fare in modo che in una oscillazione di clock il pacco si muova meno di **1 mm**. Abbiamo due opzioni:
  a) Rallentare il nastro trasportatore.
  b) Aumentare la frequenza del segnale di clock.

Esercizio 4

- Progettare **in modo *diretto*** una rete sequenziale sincrona che controlla un semaforo posto ad un attraversamento pedonale «intelligente». Il semaforo controlla il transito dei **veicoli**, permettendone il passaggio quando è verde ed impedendolo quando è rosso.



Esercizio 4

- L'incrocio è comandato da una rete logica sequenziale sincrona che riceve un clock con frequenza di 2 Hz e produce tre uscite **ROSSO**, **GIALLO** e **VERDE** che comandano le rispettive luci semaforiche. I tre segnali devono sempre essere attivi in maniera mutuamente esclusiva.

- La rete ha due ingressi **asincroni**: un segnale **PEDONE** che quando asserito codifica la presenza di pedoni in attesa di attraversare la strada e un segnale di reset **A_RESET**.

- Il funzionamento della rete prevede di mantenere il semaforo su **VERDE** se non ci sono pedoni in attesa di attraversare la strada. Questa è anche la situazione in cui viene portata la rete alla ricezione di **A_RESET.**

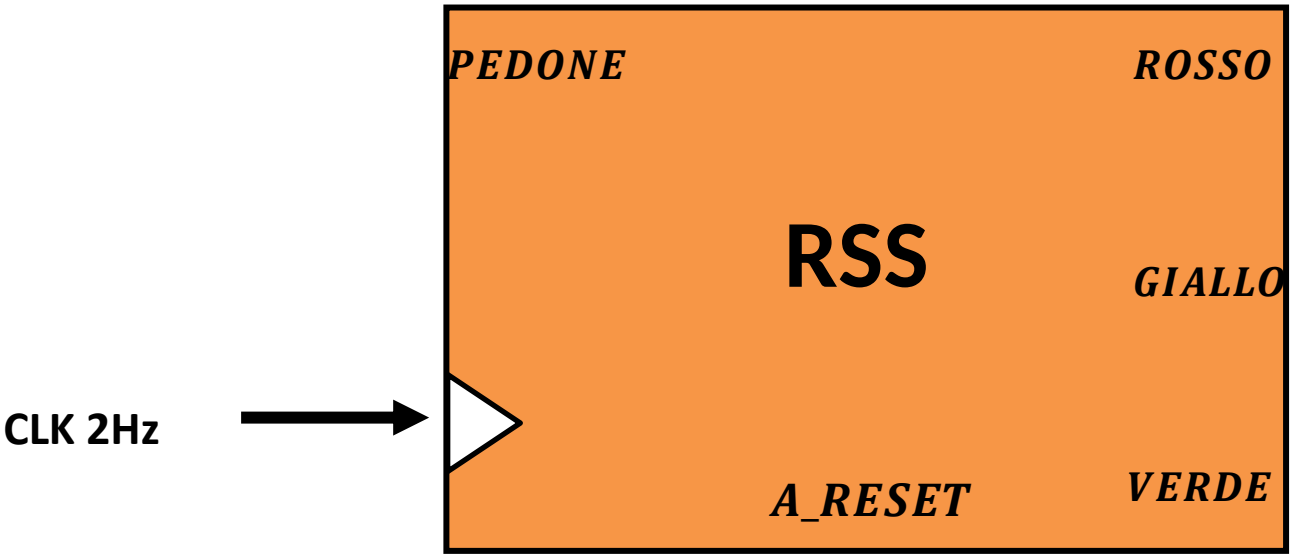
Esercizio 4

- Nel caso in cui venga rilevata la presenza di un pedone **per un periodo di almeno 1 s** la rete deve permetterne l'attraversamento accendendo **GIALLO** per **8 s** e successivamente **ROSSO** per **16 s**. Dopo questo intervallo il semaforo torna su **VERDE**, che mantiene per almeno **21 s**, **e fino alla successiva richiesta di attraversamento**.

- Nel caso in cui il semaforo completi un ciclo di attraversamento pedonale e il segnale **PEDONE** sia attivo **perché non si è mai disattivato** durante il ciclo **GIALLO-ROSSO-VERDE**, **la rete assume che il sensore sia ostruito o comunque malfunzionante** e ignora la richiesta di servizio presente, mantenendo **VERDE** attivo. Dal clock in cui **PEDONE** si disattiva, la rete riprende a considerare eventuali richieste di attraversamento provenienti dal segnale **PEDONE.**

Esercizio 4

- Si esegua la sintesi della rete minimizzando l'utilizzo delle risorse.

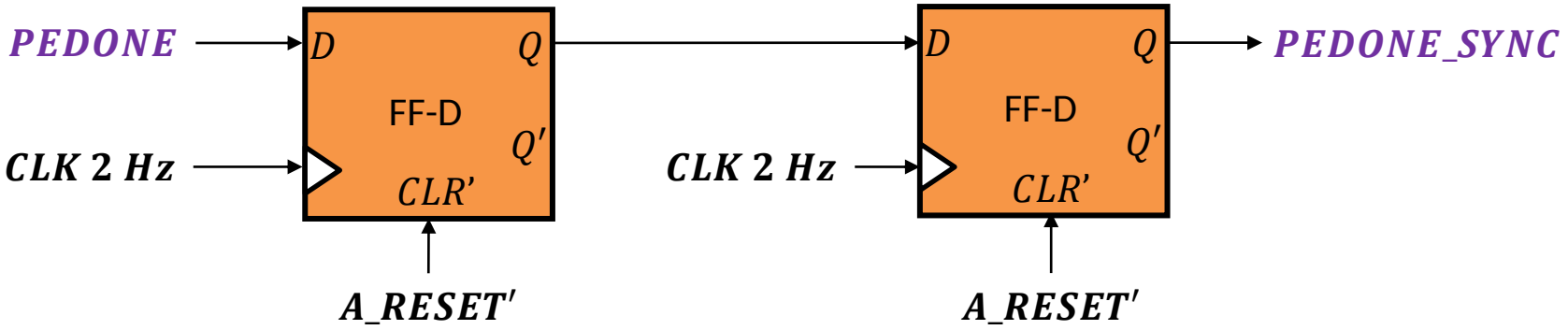


Esercizio 4 - Considerazioni

- Il segnale **PEDONE**, non essendo sincrono, deve essere opportunamente sincronizzato.
- La frequenza del clock è pari a 2 Hz, di conseguenza ogni oscillazione di clock corrisponde al passaggio di mezzo secondo.
- Possiamo attivare correttamente i segnali **GIALLO**, **ROSSO** e **VERDE** utilizzando un contatore per tenere traccia del tempo trascorso. Un intero ciclo di attraversamento pedonale dura 45 secondi (8+16+21), ovvero 90 oscillazioni di clock. Una possibile soluzione per tenere traccia delle temporizzazioni è quindi quella di utilizzare un contatore modulo 90 opportunamente comandato.
- Per rispettare la specifica sul malfunzionamento, è necessario attivare la sequenza di attraversamento in presenza di **PEDONE** a «1» da più di un secondo solo se si è visto anche un fronte di discesa (seguito da un nuovo fronte di salita) di **PEDONE** dopo l'inizio dell'ultimo attraversamento completato: questa informazione andrà memorizzata in un FF-D perché il fronte potrebbe presentarsi durante un attraversamento, ma l'informazione deve essere «usata» alla fine.

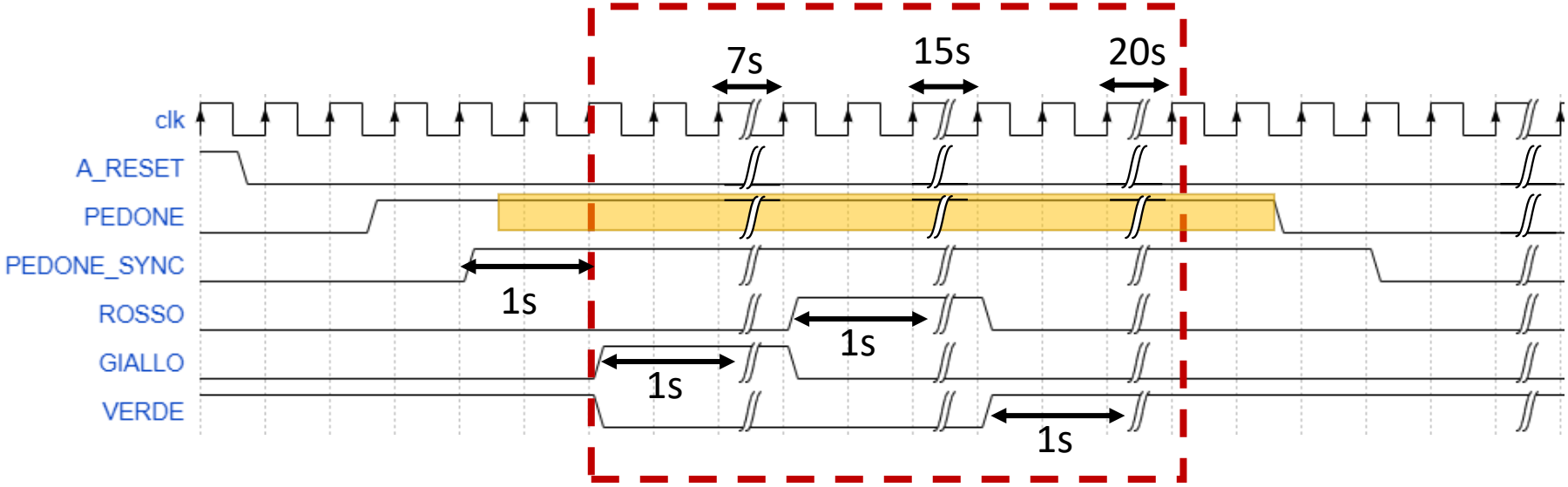
Esercizio 4 – Sincronizzazione

PEDONE può essere sincronizzato utilizzando due FFD in sequenza per generare una versione sincrona del segnale che chiameremo **PEDONE_SYNC.**



Esercizio 4 – Forme d'onda

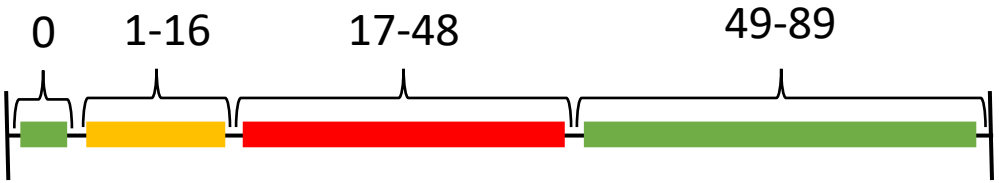
- Il comportamento che ci attendiamo dalla rete è riportato nelle forme d'onda sottostanti.
- In rosso tratteggiato indichiamo la sequenza di attraversamento pedonale fatta scattare da **PEDONE_SYNC** che assume il valore 1 per almeno due cicli di clock (1s)
- In giallo evidenziamo che queste forme d'onda identificano il caso del malfunzionamento, ovvero **PEDONE** e quindi **PEDONE_SYNC** sono ancora attivi alla fine del servizio della richiesta di attraversamento. La rete non deve iniziare un nuovo ciclo di attraversamento fino a quando non osserva un fronte di discesa di **PEDONE_SYNC**



Esercizio 4 – Sintesi Segnali

- Per la generazione della sequenza temporizzata di attraversamento pedonale **GIALLO -> ROSSO -> VERDE** possiamo utilizzare un contatore modulo 90 e decodificarne opportunamente le uscite.
- Ipotizziamo di mantenere il contatore a 0 (**EN=0**) finché non viene richiesto un attraversamento pedonale che causa l'attivazione del contatore (**EN=1).**
- Se indichiamo con **Q[6..0]** i sette bit di stato del contatore, i tre segnali di uscita possono essere ottenuti come:

$$ZERO = Q_6' Q_5' Q_4' Q_3' Q_2' Q_1' Q_0'$$



$$GIALLO = Q_6' \cdot Q_5' \cdot Q_4' \cdot (Q_3 + Q_2 + Q_1 + Q_0) + Q_6' Q_5' Q_4 Q_3' Q_2' Q_1' Q_0'$$

(Contatore tra 1 e 15 oppure =16)

$$ROSSO = Q_6' \cdot Q_5' \cdot Q_4 \cdot (Q_3 + Q_2 + Q_1 + Q_0) + Q_6' \cdot Q_5 \cdot Q_4' + Q_6' Q_5 Q_4 Q_3' Q_2' Q_1' Q_0'$$

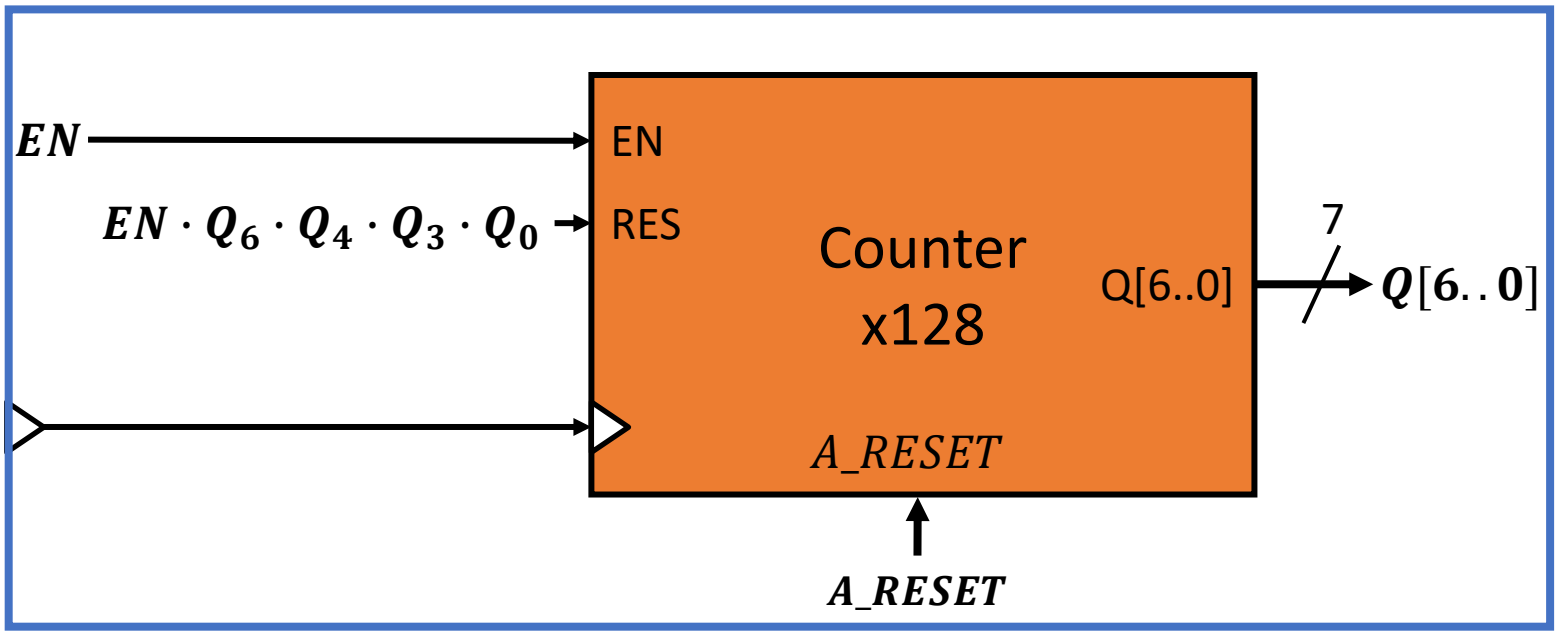
(Contatore tra 17 e 31 oppure tra 32 e 47 oppure =48)

$$VERDE = ZERO + Q_6' \cdot Q_5 \cdot Q_4 \cdot (Q_3 + Q_2 + Q_1 + Q_0) + Q_6$$

(Contatore =0 oppure tra 49 e 63 oppure >63)

Esercizio 4 – Contatore x90

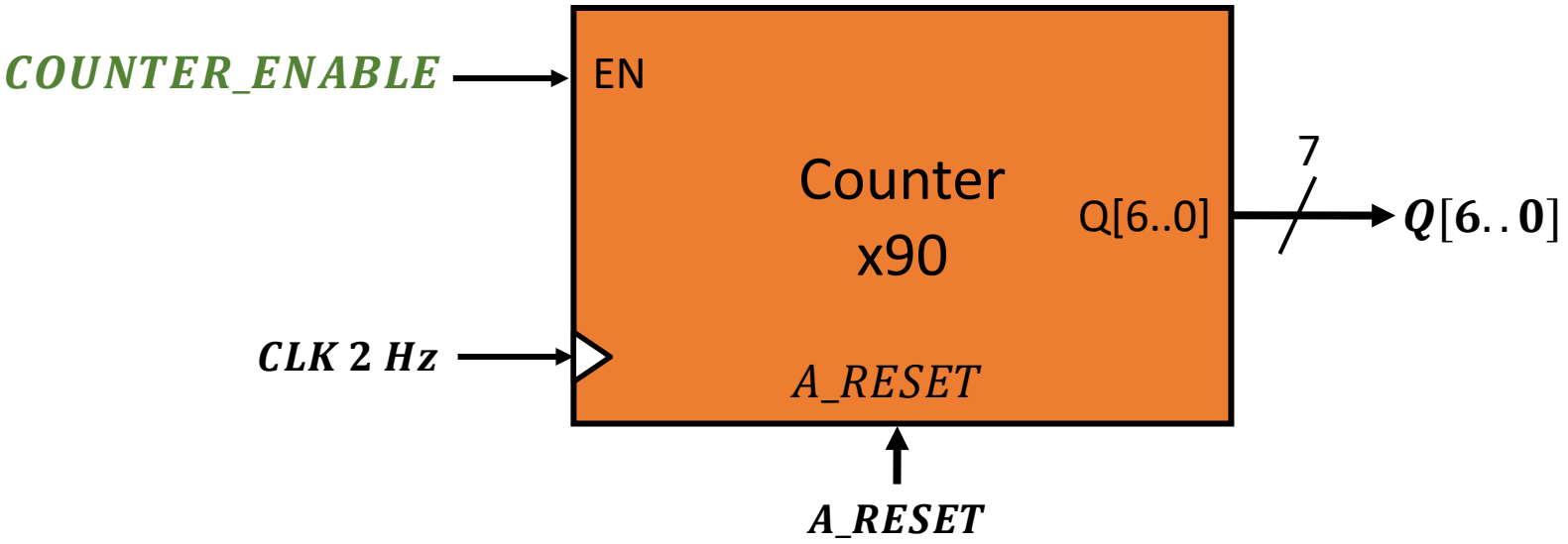
- Per realizzare un contatore x90 partiamo da un contatore x128 e ne riduciamo la base di conteggio riconoscendo il numero $(89)_{10} = (1011001)_2$



Si faccia riferimento alle slide di teoria per l'implementazione di un contatore x128 a partire da contatori x4 e x8 con segnali di **EN** e **RES**, con RES prioritario su EN.

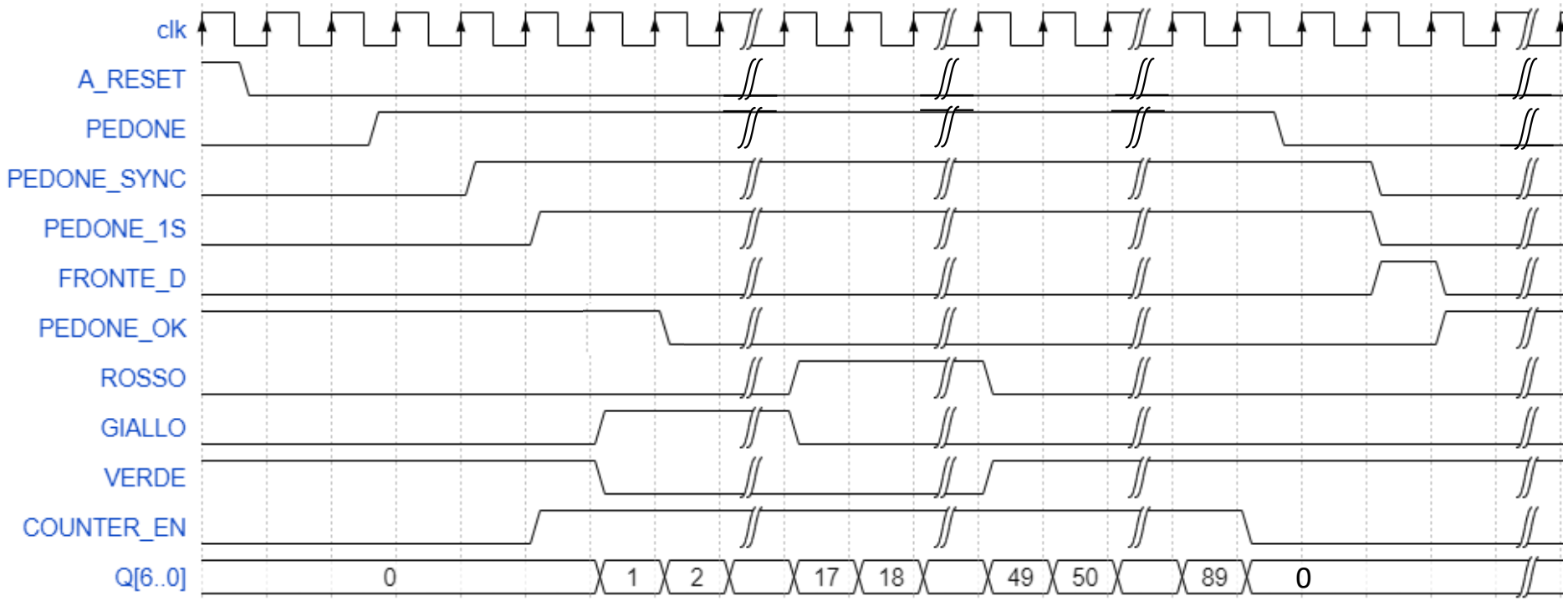
Esercizio 4 – Sintesi Segnali

- L'unico segnale da progettare è il **COUNTER_ENABLE** del contatore x90



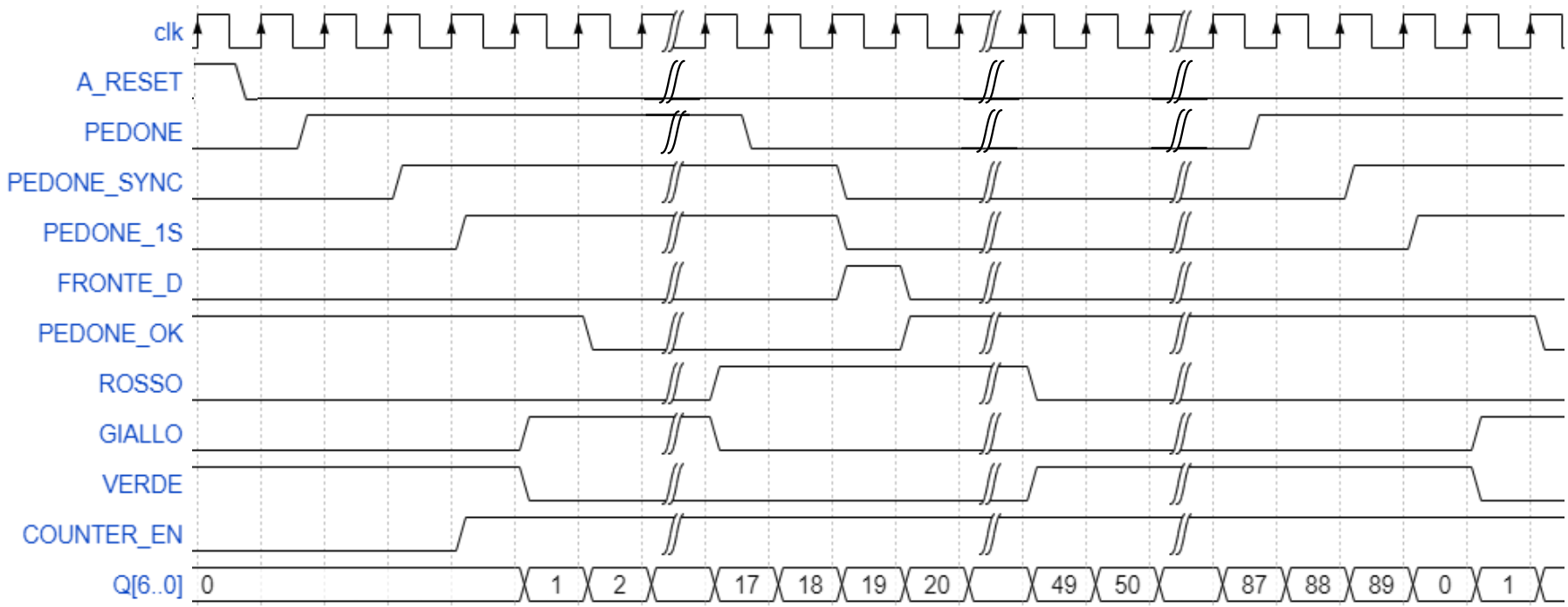
Esercizio 4 – Forme d'onda 2

- Il segnale **COUNTER_ENABLE** sincrono dovrà asserirsi quando, dall'inizio dell'ultimo attraversamento servito, c'è stato un fronte di discesa di **PEDONE_SYNC** e **PEDONE_SYNC** è ad «1» da almeno 1 secondo.
- Chiamiamo **PEDONE_OK** il segnale che memorizza se c'è stato un fronte di discesa di **PEDONE_SYNC**, e **PEDONE_1S** quello che indica in modo sincrono se **PEDONE_SYNC=1** da almeno 1 secondo.



Esercizio 4 – Forme d'onda 3

- Caso in cui ho il fronte di discesa e di salita di **PEDONE_SYNC** durante l'attraversamento.
- **PEDONE_OK** ricorda di aver visto il **FRONTE** di discesa di **PEDONE_SYNC** fino a che non inizia il servizio della richiesta.



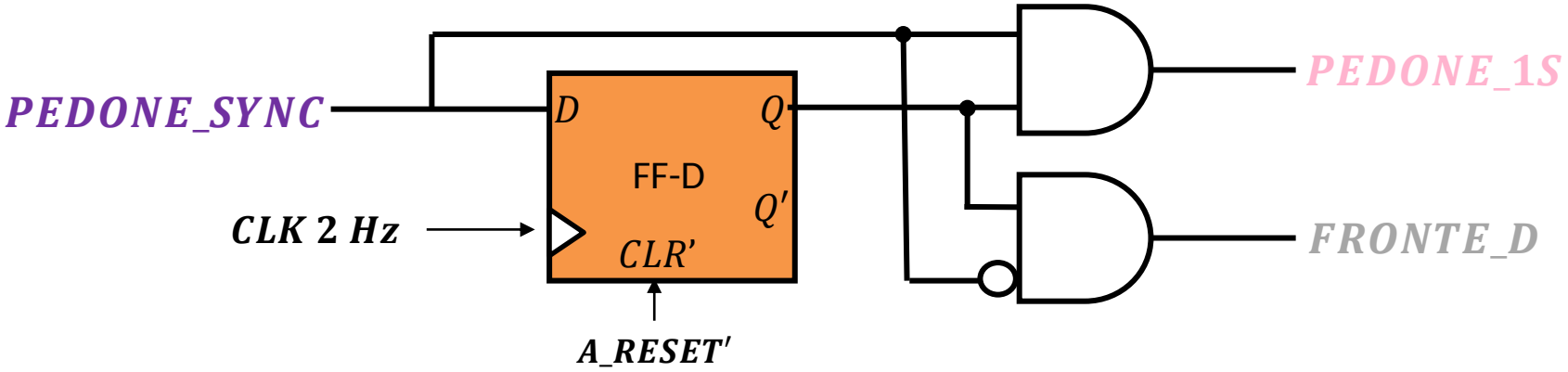
Esercizio 4 – COUNTER_ENABLE

- Il contatore deve iniziare a contare quando si trova in stato **ZERO** e si verificano entrambe le condizioni **PEDONE_OK** e **PEDONE_1S** (**COUNTER_ENABLE**=1).
- Dopo aver iniziato a contare, il contatore non deve fermarsi (**COUNTER_ENABLE**=1) fino a quando non ritorna in **ZERO**, momento nel quale:
  - Se si verificano di nuove entrambe le condizioni **PEDONE_OK** e **PEDONE_1S** , dovrà riprendere immediatamente a contare (**COUNTER_ENABLE**=1)
  - Se non si verificano tali condizioni, dovrà restare in **ZERO** (**COUNTER_ENABLE**=0)

$$COUNTER\_ENABLE = ZERO' + PEDONE\_OK \cdot PEDONE\_1S$$

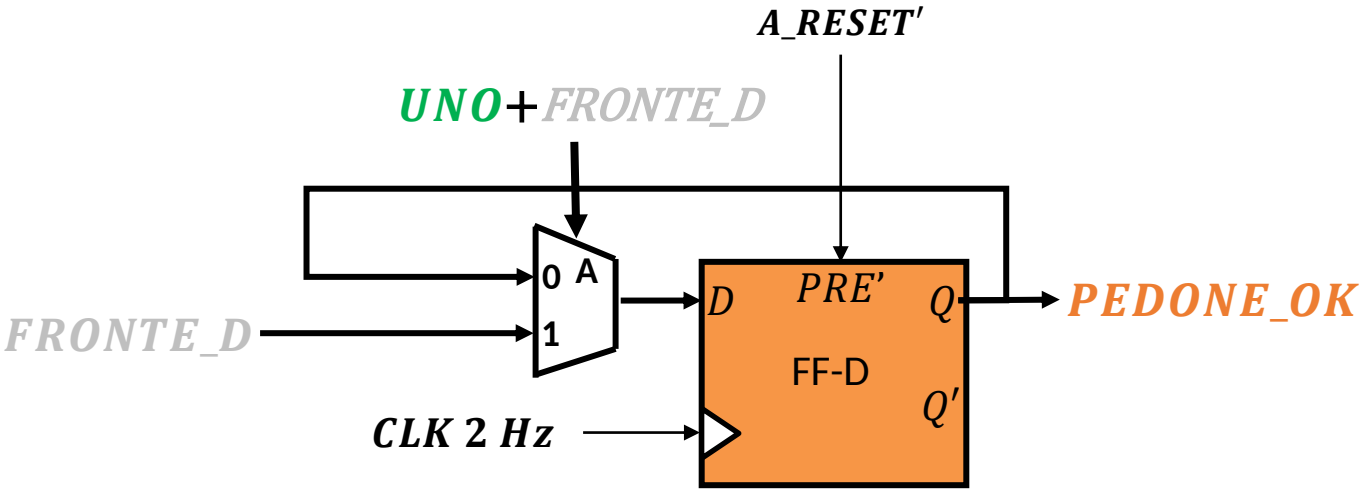
Esercizio 4 – PEDONE_1S

PEDONE_1S è un segnale che deve valere «1» quando ***PEDONE_SYNC*** vale «1» da almeno 1 secondo, ovvero nel clock precedente e in quello attuale.
Con lo stesso FF-D individuiamo anche il fronte di discesa di ***PEDONE_SYNC*** per generare il segnale ***FRONTE_D***.



Esercizio 4 – PEDONE_OK

$PEDONE\_OK$ viene memorizzato in un registro e portato a «0» ogni volta che la rete inizia un nuovo attraversamento, ovvero il contatore si trova in stato UNO, e torna ad «1» se la rete osserva un fronte di discesa di $PEDONE\_SYNC$, ovvero $FRONTE\_D$=«1».



$$UNO = Q_6' \cdot Q_5' \cdot Q_4' \cdot Q_3' \cdot Q_2' \cdot Q_1' \cdot Q_0$$